Fast and Fourier ICPC Team Notebook

Contents

1 C++

1.1 C++ template

```cpp
#include <bits/stdc++.h>

#define fi first
#define se second
#define forn(i,n) for(int i=0; i< (int)n; ++i)
#define for1(i,n) for(int i=1; i<= (int)n; ++i)
#define fore(i,l,r) for(int i=(int)l; i<= (int)r; ++i)
#define ford(i,n) for(int i=(int)(n) - 1; i>= 0; --i)
#define fored(i,l,r) for(int i=(int)r; i>= (int)l; --i)
#define pb push_back
#define el '\n'
#define d(x) cout<< #x<< " " << x<<el
#define ri(n) scanf("%d",&n)
#define sz(v) ((int)v.size())
#define all(v) v.begin(),v.end()
#define allr(v) v.rbegin(),v.rend()

using namespace std;

typedef long long ll;
typedef double ld;
typedef pair<int,int> ii;
typedef pair<char,int> pci;
typedef tuple<int, int, int> tiii;
typedef pair<ll,ll> pll;
typedef vector<int> vi;

const int inf = 1e9;
const int nax = 1e5+200;
const ld pi = acos(-1);
const ld eps= 1e-9;

int dr[] = {1,-1,0, 0,1,-1,-1, 1};
int dc[] = {0, 0,1,-1,1, 1,-1,-1};

int main(){
  ios_base::sync_with_stdio(false);
  cin.tie(NULL); cout.tie(NULL);
  cout << setprecision(20)<< fixed;
}
```

1.2 Opcion

```cpp
// En caso de que no sirva #include <bits/stdc++.h>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <sstream>
#include <fstream>
#include <cassert>
#include <climits>
#include <cstdlib>
#include <cstring>
#include <string>
#include <cstdio>
#include <vector>
#include <cmath>
#include <queue>
#include <deque>
#include <stack>
#include <list>
#include <map>
#include <set>
#include <bitset>
#include <iomanip>
#include <unordered_map>

////
#include <tuple>
#include <random>
#include <chrono>
```

1.3 Bits Manipulation

```cpp
mask |= (1<<n) // PRENDER BIT-N
mask ^= (1<<n) // FLIPPEAR BIT-N
mask &= ~(1<<n) // APAGAR BIT-N
if(mask&(1<<n)) // CHECKEAR BIT-N
T = mask&(-mask); // LSO
__builtin_ffs(mask); // INDICE DEL LSO
//  iterar sobre los subconjuntos del conjunto S
for(int subset= S; subset; subset= (subset-1) & S)
for (int subset=0;subset=subset-S&S;)    // Increasing
    order
```

1.4 Random

```cpp
// Declare number generator
mt19937 / mt19937_64 rng(chrono::steady_clock::now().
    time_since_epoch().count())
// or
random_device rd
mt19937 / mt19937_64 rng(rd())

// Use it to shuffle a vector
shuffle(permutation.begin(), permutation.end(), rng)

// Use it to generate a random number between [fr, to]
uniform_int_distribution<T> / uniform_real_distribution<T
    > dis(fr, to);
  dis(rng)

int rand(int a, int b) {
  return uniform_int_distribution<int>(a, b)(rng);
}
```

1.5 Custom Hash

```cpp
struct custom_hash {
    static ll splitmix64(ll x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(ll x) const {
        static const ll FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count()
            ;
        return splitmix64(x + FIXED_RANDOM);
    }
};
unordered_map<ll,int, custom_hash> mapa;
```

1.6 Other

```cpp
#pragma GCC optimize("O3")
//(UNCOMMENT WHEN HAVING LOTS OF RECURSIONS)\
#pragma comment(linker, "/stack:200000000")
//(UNCOMMENT WHEN NEEDED)
#pragma GCC optimize("Ofast,unroll-loops,no-stack-
    protector,fast-math")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,
    mmx,avx,tune=native")

// Custom comparator for set/map
struct comp {
        bool operator()(const double& a, const double& b)
             const {
                return a+EPS<b;}
};
set<double,comp> w; // or map<double,int,comp>
```

```
// double inf
const double DINF=numeric_limits<double>::infinity();

int main() {
  // Ouput a specific number of digits past the decimal
     point,
  // in this case 5
  // #include <iomanip>
  cout << setfill(' ') << setw(3) << 2 << endl;

  cout.setf(ios::fixed); cout << setprecision(5);
  cout << 100.0/7.0 << endl;
  cout.unsetf(ios::fixed

  // Output the decimal point and trailing zeros
  cout.setf(ios::showpoint); cout << 100.0 << endl; cout.
     unsetf(ios::showpoint);

  // Output a + before positive values
  cout.setf(ios::showpos); cout << 100 << " " << -100 <<
     endl; cout.unsetf(ios::showpos);

  // Output numerical values in hexadecimal
  cout << hex << 100 << " " << 1000 << " " << 10000 <<
     dec << endl;
}
```

2 Strings

2.1 Z's Algorithm

```
// O(|s|)
vi z_function(string &s){
  int n = s.size();
  vi z(n);
  int x = 0, y = 0;
  for(int i = 1; i < n; ++i) {
    z[i] = max(0, min(z[i-x], y-i+1));
    while (i+z[i] < n && s[z[i]] == s[i+z[i]])
      x = i, y = i+z[i], z[i]++;
  }
  return z;
}
```

2.2 KMP

```
vi get_phi(string &s) { // O(|s|)
  int j = 0, n = sz(s); vi pi(n);
  for1(i,n-1){
    while (j > 0 && s[i] != s[j]) j = pi[j-1];
    j += (s[i] == s[j]);
    pi[i] = j;
  }
  return pi;
}
void kmp(string &t, string &p){ // O(|t| + |p|)
  vi phi = get_phi(p);
  int matches = 0;
  for(int i = 0, j = 0; i < sz(t); ++i ) {
    while(j > 0 && t[i] != p[j] ) j = phi[j-1];
    if(t[i] == p[j]) ++j;
    if(j == sz(p)) {
      matches++;
      j = phi[j-1];
    }
  }
}
/// Automaton
/// Complexity O(n*C) where C is the size of the alphabet
int aut[nax][26];
void kmp_aut(string &p) {
  int n = sz(p);
  vi phi = get_phi(p);
  forn(i, n+1) {
    forn(c, 26) {
      if (i==n || (i>0 && 'a'+c!= p[i])) aut[i][c] = aut[
          phi[i-1]][c];
      else  aut[i][c] = i + ('a'+c == p[i]);
    }
  }
}

/// Automaton
int wh[nax+2][MAXC];    //wh[i][j] = a donde vuelvo si
    estoy en i y pongo una j
void build(string &s){
        int lps=0;
        wh[0][s[0]-'a'] = 1;
        fore(i,1,sz(s)){
                fore(j,0,MAXC-1) wh[i][j]=wh[lps][j];
                if(i<sz(s)){
                        wh[i][s[i]-'a'] = i+1;
                        lps = wh[lps][s[i]-'a'];
                }
        }
}
```

2.3 Hashing

```
/// 1000234999, 1000567999, 1000111997, 1000777121,
    999727999, 1070777777
const int MODS[] = { 1001864327, 1001265673 };
const ii BASE(257, 367), ZERO(0, 0), ONE(1, 1);
inline int add(int a, int b, const int& mod) { return a+b
```

```
    >= mod ? a+b-mod : a+b; }
inline int sbt(int a, int b, const int& mod) { return a-b
    < 0 ? a-b+mod : a-b; }
inline int mul(int a, int b, const int& mod) { return 1ll
   *a*b%mod; }
inline ll operator ! (const ii a) { return (ll(a.fi)<<32)
   |ll(a.se); }
inline ii operator + (const ii a, const ii b) {
  return {add(a.fi, b.fi, MODS[0]), add(a.se, b.se, MODS
     [1])};
}
inline ii operator - (const ii a, const ii b) {
  return {sbt(a.fi, b.fi, MODS[0]), sbt(a.se, b.se, MODS
     [1])};
}
inline ii operator * (const ii a, const ii b) {
  return {mul(a.fi, b.fi, MODS[0]), mul(a.se, b.se, MODS
     [1])};
}

const int nax = 1e5+20;
ii base[nax];
void prepare() {
  base[0] = ONE;
  for1(i,nax-1) base[i] = base[i-1]*BASE;
}
template <class type>
struct hashing {   /// HACELEEE PREPAREEEE!!!
  vector<ii> code;
  hashing(type &t) {
    code.resize(sz(t)+1);
    code[0] = ZERO;
    for1(i,sz(t))
      code[i] = code[i-1]*BASE + ii{t[i-1], t[i-1]};
  }
  ii query(int l, int r) { ///  [l,r]
    return code[r+1] - code[l]*base[r-l+1];
  }
};
```

2.4 Manacher Algorithm

```
// f = 1 para pares, 0 impar
//a a a a a a
//1 2 3 3 2 1   f = 0 impar
//0 1 2 3 2 1   f = 1 par
void manacher(string &s, int f, vi &d){
  int l=0, r=-1, n=sz(s);
  d.assign(n,0);
  forn(i, n){
    int k=(i>r? (1-f) : min(d[l+r-i+ f], r-i+f)) + f;
    while(i+k-f<n && i-k>=0 && s[i+k-f]==s[i-k]) ++k;
    d[i] = k - f; --k;
    if(i+k-f > r) l=i-k, r=i+k-f;
  }
//  forn(i,n) d[i] = (d[i]-1+f)*2 + 1-f;
}
```

2.5 Minimum Expression

```
int minExp(string &t) {
  int i = 0, j = 1, k = 0, n = sz(t), x, y;
  while (i < n && j < n && k < n) {
    x = i+k;
    y = j+k;
    if (x >= n) x -= n;
    if (y >= n) y -= n;
    if (t[x] == t[y]) ++k;
    else if (t[x] > t[y]) {
      i = j+1 > i+k+1 ? j+1 : i+k+1;
      swap(i, j);
      k = 0;
    } else {
      j = i+1 > j+k+1 ? i+1 : j+k+1;
      k = 0;
    }
  }
  return i;
}
```

2.6 Trie

```
const static int N = 1<<21;
const static int alpha = 26;
int trie[N][alpha], cnt[N], sz;
void init(){ memset(trie[0], 0 ,sizeof trie[0]); sz = 0;
   }
void add(const string &s){
  int v= 0;
  for(char ch: s){
    int c = ch-'a';
    int &next = trie[v][c];
    if(!next){
      next = ++sz;
      memset(trie[next], 0, sizeof trie[next]);
    }
    v= next;
  }
  cnt[v]++;
}
```

2.7 Suffix Array

```
struct SuffixArray { // test line 11
        vi sa, lcp;
        SuffixArray(string& s, int lim=256){
                int n = sz(s) + 1, k = 0, a, b;
                s.pb('$');
                vi x(all(s)), y(n), ws(max(n, lim)), rank
                    (n);
                sa = lcp = y, iota(all(sa), 0);
                for (int j = 0, p = 0; p < n; j = max(1,
                    j * 2), lim = p) {
                        p = j;
//      iota(all(y), n - j);
//                      forn(i,n) if (sa[i] >= j) y[p++]
    = sa[i] - j;
                        forn(i,n) y[i] = (sa[i] - j >= 0
                            ? 0 : n) + sa[i]-j; // this
                            replace the two lines
                        // before hopefully xd
                        fill(all(ws), 0);
                        forn(i,n) ws[x[i]]++;
                        for1(i,lim-1) ws[i] += ws[i - 1];
                        for (int i = n; i--;) sa[--ws[x[y
                            [i]]]] = y[i];
                        swap(x, y), p = 1, x[sa[0]] = 0;
                        for1(i,n-1) a = sa[i - 1], b = sa
                            [i], x[b] =
                                (y[a] == y[b] && y[a + j]
                                    == y[b + j]) ? p - 1
                                  : p++;
                }
                for1(i,n-1) rank[sa[i]] = i;
                for (int i = 0, j; i < n - 1; lcp[rank[i
                    ++]] = k) // lcp(i): lcp suffix i-1,i
                        for (k && k--, j = sa[rank[i] -
                            1];
                                        s[i + k] == s[j +
                                            k]; k++);
        }
};
```

2.8 Aho-Corasick

```
const static int N = 1<<21;
const static int alpha = 26;
int trie[N][alpha], fail[N], nodes, end_word[N], cnt_word
    [N], fail_out[N];
inline int conv(char ch) { // Function for properly index
     characters
  return ((ch >= 'a' && ch <= 'z') ? ch-'a' : ch-'A'+26);
}
void add(string &s, int i) {
  int cur = 0;
  for(char c : s) {
    int x = conv(c);
    if(!trie[cur][x]) trie[cur][x] = ++nodes;
    cur = trie[cur][x];
  }
  ++cnt_word[cur];
  end_word[cur] = i; // for i > 0
}
void build() { // HACELEEE build!!!!
  queue<int> q; q.push(0);
  while(sz(q)) {
    int u = q.front(); q.pop();
    for(int i = 0; i < alpha; ++i) {
      int v = trie[u][i];
      if(!v) trie[u][i] = trie[ fail[u] ][i]; //
          construir automata
      else q.push(v);
      if(!u || !v) continue;
      fail[v] = trie[ fail[u] ][i];
      fail_out[v] = end_word[ fail[v] ] ? fail[v] :
          fail_out[ fail[v] ];
      cnt_word[v] += cnt_word[ fail[v] ]; // obtener
          informacion del fail_padre
    }
  }
}
```

2.9 Suffix Automaton

```
struct node {
  int len, link;
  map<char, int> to;
  bool terminal;
};

const int nax = 1<<21;
node st[nax];
int sz, last;
int occ[nax];

void sa_init() { ///// HACELEE INIT!!!
  forn(i,sz) st[i] = node();
  st[0].len = last = st[0].terminal = 0;
  st[0].link = -1;
  sz=1;
}

void sa_extend(char c) {
  int cur = sz++;
  st[cur].len = st[last].len + 1;
  int p = last;
  while (p != -1 && !st[p].to.count(c)) {
    st[p].to[c] = cur;
    p = st[p].link;
  }
```

```
    if (p == -1)  st[cur].link = 0;
    else {
      int q = st[p].to[c];
      if (st[p].len + 1 == st[q].len) st[cur].link = q;
      else {
        int w = sz++;
        st[w].len = st[p].len + 1;
        st[w].to = st[q].to;
        st[w].link = st[q].link;
        while (p != -1 && st[p].to[c] == q) {
          st[p].to[c] = w;
          p = st[p].link;
        }
        st[q].link = st[cur].link = w;
      }
    }
    last = cur;
  }

  void dfs(int v){
    if(occ[v] != 0) return;
    occ[v] = st[v].terminal;
    for(auto &e : st[v].to){
      dfs(e.se);
      occ[v] += occ[e.se];
    }
  }

  string lcs(string &S, string &T){
    sa_init();
    for (char c : S) sa_extend(c);

    int v = 0, l = 0, best = 0, bestpos = 0;
    forn(i, sz(T)){
      while (v && !st[v].to.count(T[i])) {
        v = st[v].link , l = st[v].len ;
      }
      if (st[v].to.count(T[i])) {
        v = st[v].to[T[i]];
        l++;
      }
      if (l > best) {
        best = l;
        bestpos = i;
      }
    }
    // best guarda el tamano del longest common substring
    return T.substr(bestpos - best + 1, best);
  }
```

2.10 Palindromic Tree

```
  struct palindromic_tree{
    static const int SIGMA = 26;
    struct node{
      int link, len, p, to[SIGMA];
      node(int len, int link=0,int p=0):
            len(len),link(link),p(p){
        memset(to,0,sizeof(to));
      }
    };
    int last;
    vector<node> st;
    palindromic_tree():last(0){fore(i,-1,0)st.pb(node(i));}

    void add(int i, const string &s){
      int c = s[i]-'a';
      int p = last;
      while(s[i-st[p].len-1]!=s[i]) p=st[p].link;

      if(st[p].to[c]){
        last = st[p].to[c];
      }else{
        int q=st[p].link;
        while(s[i-st[q].len-1]!=s[i]) q=st[q].link;
        q=max(1,st[q].to[c]);
        last = st[p].to[c] = sz(st);
        st.pb(node(st[p].len+2,q,p));
      }
    }
  };
```

3 Graph algorithms

3.1 Articulation Points and Bridges

```
  // Complexity: V + E
  // Given an undirected graph
  int n, timer, tin[nax], low[nax];
  vi g[nax]; // adjacency list of graph

  void dfs(int u, int p) {
    tin[u] = low[u] = ++timer;
    int children=0;
    for (int v : g[u]) {
      if (v == p) continue;
      if (tin[v]) low[u] = min(low[u], tin[v]);
      else {
        dfs(v, u);
        low[u] = min(low[u], low[v]);
        if (low[v] > tin[u]) // BRIDGE
          IS_BRIDGE(u, v);

        if (low[v] >= tin[u] && p!=-1) // POINT
          IS_CUTPOINT(u);
        ++children;
      }
    }
    if(p == -1 && children > 1) // POINT
```

```
        IS_CUTPOINT(u);
  }

  void find_articulations() {
    timer = 0;
    forn(i,n) if(!tin[i]) dfs(i,-1);
  }
```

3.2 Biconnected Components

```
struct edge {
        int u, v, comp; //A que componente biconexa
            pertenece
        bool bridge; //Si la arista es un puente
};

vector<int> g[nax]; //Lista de adyacencia
vector<edge> e; //Lista de aristas
stack<int> st;
int low[nax], num[nax], cont;
int art[nax]; //Si el nodo es un punto de articulacion
//vector<vector<int>> comps; //Componentes biconexas
//vector<vector<int>> tree; //Block cut tree
//vector<int> id; //Id del nodo en el block cut tree
int nbc; //Cantidad de componentes biconexas
int N, M; //Cantidad de nodos y aristas

void add_edge(int u, int v){
        g[u].pb(sz(e)); g[v].pb(sz(e));
        e.pb({u, v, -1, false});
}
void dfs(int u, int p = -1) {
        low[u] = num[u] = cont++;
        for (int i : g[u]) {
                edge &ed = e[i];
                int v = ed.u^ed.v^u;
                if(num[v]<0){
                        st.push(i);
                        dfs(v, i);
                        if (low[v] > num[u]) ed.bridge =
                            true; //bridge
                        if (low[v] >= num[u]) {
                                art[u]++; //articulation
                                int last; //start
                                    biconnected
//                              comps.pb({});
                                do {
                                        last = st.top();
                                            st.pop();
                                        e[last].comp =
                                            nbc;
//                                      comps.back().pb(e
    [last].u);
//                                      comps.back().pb(e
    [last].v);
                                } while (last != i);
                                nbc++; //end biconnected
                        }
                        low[u] = min(low[u], low[v]);
                } else if (i != p && num[v] < num[u]) {
                        st.push(i);
                        low[u] = min(low[u], num[v]);
                }
        }
}

void build_tree() {
        tree.clear(); id.resize(N); tree.reserve(2*N);
        forn(u,N)
                if (art[u]) id[u] = sz(tree); tree.pb({})
                    ;
        for (auto &comp : comps) {
    sort(all(comp));
    comp.resize(unique(all(comp)) - comp.begin());
                int node = sz(tree);
                tree.pb({});
                for (int u : comp) {
                        if (art[u]) {
                                tree[id[u]].pb(node);
                                tree[node].pb(id[u]);
                        }else id[u] = node;
                }
        }
}
void doit() {
        cont = nbc = 0;
//      comps.clear();
        forn(i,N) {
                g[i].clear(); num[i] = -1; art[i] = 0;
        }
        forn(i,N){
    if(num[i]<0) dfs(i), --art[i];
        }
}
```

3.3 Topological Sort

```
vi g[nax], ts;
bool seen[nax];
void dfs(int u){
  seen[u] = true;
  for(int v: g[u])
    if (!seen[v])
      dfs(v);
  ts.pb(u);
}
void topo(int n){
  forn(i,n) if (!seen[i]) dfs(i);
```

```
  reverse(all(ts));
}
```

3.4 Kosaraju: Strongly connected components

```
vi g[nax], gr[nax], ts;
int scc[nax];
bool seen[nax];
void dfs1(int u){
  seen[u] = true;
  for (int v: g[u])
    if (!seen[v])
      dfs1(v);
  ts.pb(u);
}
void dfs(int u, int comp){
  scc[u] = comp;
  for (int v : gr[u])
    if (scc[v] == -1)
      dfs(v, comp);
}
// Retorna la cantidad de componentes
int find_scc(int n){
  //TENER CREADO EL GRAFO REVERSADO gr
  forn(i,n) if(!seen[i]) dfs1(i);
  reverse(all(ts));
  int comp = 0;
  for(int u: ts)
    if (scc[u] == -1) dfs(u, comp++);
  return comp;
}
```

3.5 Tarjan: Strongly connected components

```
vi low, num, comp, g[nax];
int scc, timer;
stack<int> st;
void tjn(int u) {
  low[u] = num[u] = timer++; st.push(u); int v;
  for(int v: g[u]) {
    if(num[v]==-1) tjn(v);
    if(comp[v]==-1) low[u] = min(low[u], low[v]);
  }
  if(low[u]==num[u]) {
    do{ v = st.top(); st.pop(); comp[v]=scc;
    }while(u != v);
    ++scc;
  }
}
void callt(int n) {
  timer = scc= 0;
  num = low = comp = vector<int>(n,-1);
  forn(i,n) if(num[i]==-1) tjn(i);
}
```

3.6 MST Kruskal

```
// Complejidad O( E* log V)
struct edge{
  int u, v, w;
  edge(int u, int v, int w):u(u),v(v),w(w){}
  bool operator < (const edge &o) const{
    return w < o.w;//if want max, change this
  }
};
vector<edge> g, st;
dsu uf; // union-find
int kruskal(int n){
  uf.init(n);
  sort(all(g));
  int total = 0, u, v, w;
  for(int i = 0; i < sz(g) && uf.numSets!=1; ++i){
    u = g[i].u;
    v = g[i].v;
    w = g[i].w;
    if(!uf.isSameSet(u,v)){
      total+= w;
      uf.unionSet(u,v);
      st.pb(g[i]);
    }
  }
  return total;
}
```

3.7 MST Prim

```
//Complexity O(E * log V)
vector<ii> g[nax];
bool seen[nax];
priority_queue<ii> pq;
void process(int u) {
  seen[u] = true;
  for (ii v: g[u])
    if (!seen[v.fi])
      pq.push(ii(-v.se, v.fi));
}
int prim(int n){
  process(0);
  int total = 0, u, w;
  while (sz(pq)){
    ii e = pq.top(); pq.pop();
    tie(w,v) = e; w*=-1;
    if (!seen[u])
      total += w, process(u);
```

```
  }
  return total;
}
```

3.8 Dijkstra

```
// O ((V+E)*log V)
vector <ii> g[nax];
int d[nax], p[nax];
void dijkstra(int s, int n){
  forn(i,n) d[i] = inf, p[i] = -1;
  priority_queue <ii, vector <ii>,greater<ii> > q;
  d[s] = 0;
  q.push(ii(0, s));
  int dist, u, v, w;
  while(sz(q)){
    tie(dist, u) = q.top();
    q.pop();
    if (dist > d[u]) continue;
    for (ii e: g[u]){
      tie(v,w) = e;
      if (d[u] + w < d[v]){
        d[v] = d[u] + w;
        p[v] = u;
        q.push(ii(d[v], v));
      }
    }
  }
}
vi find_path(int t){
  vi path;
  int cur = t;
  while(cur != -1){
    path.pb(cur);
    cur = p[cur];
  }
  reverse(all(path));
  return path;
}
```

3.9 Bellman-Ford

```
// O(E*V) list adjacency, O(V^3) matrix
vector<ii> g[nax];
int dist[nax];
void bellman_ford(int s, int n){
  forn(i,n) dist[i] = inf;
  dist[s] = 0; int v, w;
  forn(i,n-1){
    forn(u,n){
      for(ii e : g[u]){
        tie(v,w) = e;
        dist[v] = min(dist[v], dist[u] + w);
      }
    }
  }
  bool negcycle= false;
  forn(u,n){
    for(ii e: g[u]){
      tie(v,w) = e;
      if (dist[v] > dist[u] + w) negcycle = true;
    }
  }
}
```

3.10 Shortest Path Faster Algorithm

```
// Complexity O(V*E) worst, O(E) on average.
vector<ii> g[nax];
bool inqueue[nax];
int n;
bool spfa(int s, vi& dist) {
  dist.assign(n, inf);
  vi cnt(n, 0);
  queue<int> q;
  dist[s] = 0;
  q.push(s);
  inqueue[s] = true;
  int u, v, w;
  while(sz(q)) {
    u = q.front(); q.pop();
    inqueue[u] = false;
    for (ii e: g[u]) {
      tie(v, w) = e;
      if (dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w;
        if (!inqueue[v]) {
          q.push(v);
          inqueue[v] = true;
          cnt[v]++;
          if (cnt[v] > n) return false;  // negative
              cycle
        }
      }
    }
  }
  return true;
}
```

3.11 Floyd-Warshall

```
// Complejidad O(n^3)
int dist[nax][nax];
void floyd(){
```

```
    // Hay que saber inicializar el array d.
    forn(k,n){
      forn(u,n){
        forn(v,n){
          dist[u][v] = min(dist[u][v], dist[u][k] + dist[k
              ][v]);
        }
      }
    }
  }
```

3.12 LCA Binary Lifting

```
const int L = 24;
int timer, up[nax][L+1], n;
int in[nax], out[nax];
vi g[nax];
void dfs(int u, int p){
  in[u] = ++timer;
  up[u][0] = p;
  for1(i,L) up[u][i] = up[up[u][i-1]][i-1];
  for(int v: g[u]){
    if(v==p) continue;
    dfs(v,u);
  }
  out[u] = ++timer;
}
bool anc(int u, int v){
  return in[u]<= in[v] && out[u]>= out[v];
}
void solve(int root){
  timer = 0;
  dfs(root,root);
}
int lca(int u, int v){
  if(anc(u,v)) return u;
  if(anc(v,u)) return v;
  for(int i= L; i>=0; --i){
    if(!anc(up[u][i],v))
      u = up[u][i];
  }
  return up[u][0];
}
```

3.13 2 SAT

```
// Complexity O(V+E)
int N;
vi low, num, comp, g[nax];
vector<bool> truth;
int scc, timer;
stack<int> st;
void tjn(int u) {
  low[u] = num[u] = timer++; st.push(u); int v;
  for(int v: g[u]) {
    if(num[v]==-1) tjn(v);
    if(comp[v]==-1) low[u] = min(low[u], low[v]);
  }
  if(low[u]==num[u]) {
    do{ v = st.top(); st.pop(); comp[v]=scc;
    }while(u != v);
    ++scc;
  }
}
bool solve_2SAT() {
  int n = 2*N;
  timer = scc= 0;
  num = low = comp = vi(n,-1);
  forn(i,n)
    if(num[i]==-1) tjn(i);
  truth = vector<bool>(N, false);
  forn(i,N) {
    if (comp[i] == comp[i + N]) return false;
    truth[i] = comp[i] < comp[i + N];
  }
  return true;
}
int neg(int x){
  if(x<N) return x+N;
  else return x-N;
}
void add_edge(int x, int y){
  g[x].pb(y);
}
void add_disjuntion(int x, int y){
  add_edge(neg(x), y);
  add_edge(neg(y), x);
}
void implies(int x, int y) {
  add_edge(x,y);
  add_edge(neg(y),neg(x));
}
void make_true(int u) { add_edge(neg(u), u); }
void make_false(int u) { make_true(neg(u)); }
void make_eq(int x, int y){
  implies(x, y);
  implies(y, x);
}
void make_dif(int x, int y){
  implies(neg(x), y);
  implies(neg(y), x);
}
```

3.14 Centroid Decomposition

```
int cnt[nax], depth[nax], f[nax], dist[25][nax];
```

```
vi g[nax];
int dfs(int u, int dep = -1, bool flag = 0, int dis = 0,
    int p = -1) {
  cnt[u] = 1;
  if(flag) dist[dep][u] = dis;
  for (int v : g[u])
    if (!depth[v] && v != p) cnt[u] += dfs(v, dep, flag,
        dis + 1, u);
  return cnt[u];
}
int get_centroid (int u, int r, int p = -1) {
  for (int v : g[u])
    if (!depth[v] && v != p && cnt[v] > r)
      return get_centroid(v, r, u);
  return u;
}
int decompose(int u, int d = 1) {
  int centroid = get_centroid(u, dfs(u)>>1);
  depth[centroid] = d;
  dfs(centroid, d); /// if distances is needed
  for (int v : g[centroid])
    if (!depth[v])
      f[decompose(v, d + 1)] = centroid;
  return centroid;
}
int lca (int u, int v) {
  for (; u != v; u = f[u])
    if (depth[v] > depth[u])
      swap(u, v);
  return u;
}
int get_dist(int u, int v){
  int dep_l = depth[lca(u,v)];
  return dist[dep_l][u] + dist[dep_l][v];
}
```

3.15 Tree Binarization

```
vi g[nax];
int son[nax], bro[nax];
void binarize(int u, int p = -1){
  bool flag = 0; int prev = 0;
  for(int v : g[u]){
    if(v == p) continue;
    if(flag) bro[prev] = v;
    else son[u] = v, flag = true;
    binarize(v, u);
    prev = v;
  }
}
```

3.16 Eulerian Path

```
int n;
int edges = 0;
int out[nax], in[nax];

// Directed version (uncomment commented code for
   undirected)
struct edge {
        int v;
//      list<edge>::iterator rev;
        edge(int v):v(v){}
};
list<edge> g[nax];
void add_edge(int a, int b){
  out[a]++;
  in[b]++;
        ++edges;
        g[a].push_front(edge(b));//auto ia=g[a].begin();
//      g[b].push_front(edge(a));auto ib=g[b].begin();
//      ia->rev=ib;ib->rev=ia;
}
vi p;
void go(int u){
        while(sz(g[u])){
                int v=g[u].front().v;
                //g[v].erase(g[u].front().rev);
                g[u].pop_front();
                go(v);
        }
        p.push_back(u);
}

vi get_path(int u){
        p.clear();
        go(u);
        reverse(all(p));
        return p;
}

/// for undirected uncomment and check for path existance
bool eulerian(vi &tour) { /// directed graph
  int one_in = 0, one_out = 0, start = -1;
  bool ok = true;
  for (int i = 0; i < n; i++) {
    if(out[i] && start == -1) start = i;
    if(out[i] - in[i] == 1) one_out++, start = i;
    else if(in[i] - out[i] == 1) one_in++;
    else ok &= in[i] == out[i];
  }
  ok &= one_in == one_out && one_in <= 1;
  if (ok) {
    tour = get_path(start);
    if(sz(tour) == edges + 1) return true;
  }
  return false;
}
```

4 Flows

4.1 Edmons-Karp

```
// Complexity O(V*E^2)
const ll inf = 1e18;

struct EKarp{
  vector<int> q, dist, p;
  vector<vector<ll>> cap, flow;
  vector<vector<int>> g;
  int n, s, t;

  EKarp(int n_){
    n = n_; g.resize(n);
    cap = flow = vector<vector<ll>>(n,vector<ll>(n));
  }

  void addEdge(int u, int v, ll c){
    cap[u][v] = c;
    g[u].pb(v); g[v].pb(u);
  }

  ll bfs(int s, int t) {
    p.assign(n, -1);  p[s] = -2;
    queue<pair<int,ll>> q;
    q.push(pair<int,ll>(s, inf));
    while (!q.empty()) {
      int u = q.front().fi; ll f = q.front().se;
      q.pop();
      for(int v: g[u]){
        if (p[v] == -1 && cap[u][v] - flow[u][v]>0) {
          p[v] = u;
          ll df = min(f, cap[u][v]-flow[u][v]);
          if (v == t) return df;
          q.push(pair<int,ll>(v, df));
        }
      }
    }
    return 0;
  }
  ll maxFlow() {
    ll mf = 0;
    ll f;
    while (f = bfs(s,t)){
      mf += f;
      int v = t;
      while (v != s) {
        int prev = p[v];
        flow[v][prev] -= f;
        flow[prev][v] += f;
        v = prev;
      }
    }
    return mf;
  }
};
```

4.2 Dinic

```
// Corte minimo: vertices con dist[v]>=0 (del lado de src
    ) VS.  dist[v]==-1 (del lado del dst)
// Para el caso de la red de Bipartite Matching (Sean V1
    y V2 los conjuntos mas proximos a src y dst
    respectivamente):
// Reconstruir matching: para todo v1 en V1 ver las
    aristas a vertices de V2 con it->f>0, es arista del
    Matching
// Min Vertex Cover: vertices de V1 con dist[v]==-1 +
    vertices de V2 con dist[v]>0
// Max Independent Set: tomar los vertices NO tomados por
     el Min Vertex Cover
// Max Clique: construir la red de G complemento (debe
    ser bipartito!) y encontrar un Max Independet Set
// Min Edge Cover: tomar las aristas del matching + para
    todo vertices no cubierto hasta el momento, tomar
    cualquier arista de el

// Complexity O(V^2*E)
const ll inf = 1e18;

struct Dinic{
  struct edge {
    int to, rev; ll f, cap;
    edge(int to, int rev, ll cap, ll f=0) : to(to), rev(
        rev), f(f), cap(cap) {}
  };
  vector<vector<edge>> g;
  vector<int> q, dist, work;
  int n, s, t;

  Dinic(int n_){
    n = n_; g.resize(n);
    q.resize(n);
  }

  void addEdge(int s, int t, ll cap){
      g[s].pb(edge(t, sz(g[t]), cap));
      g[t].pb(edge(s, sz(g[s])-1, 0));
  }

  bool bfs(){
    dist.assign(n,-1), dist[s]=0;
    int qt=0;
    q[qt++]=s;
    for(int qh=0; qh<qt; ++qh){
      int u =q[qh];
      for(edge e: g[u]){
        int v=e.to;
```

```
        if(dist[v]<0 && e.f < e.cap)
          dist[v]=dist[u]+1, q[qt++]=v;
      }
    }
    return dist[t]>=0;
  }

  ll dfs(int u, ll f){
    if(u==t) return f;
    for(int &i=work[u]; i<sz(g[u]); ++i){
      edge &e = g[u][i];
      if(e.cap<=e.f) continue;
      int v=e.to;
      if(dist[v]==dist[u]+1){
        ll df=dfs(v, min(f, e.cap-e.f));
        if(df>0){
          e.f+=df, g[v][e.rev].f-= df;
          return df;
        }
      }
    }
    return 0;
  }

  ll maxFlow(int s_, int t_){
    s = s_, t = t_;
    ll max_flow=0;
    while(bfs()){
      work.assign(n,0);
      while(ll delta=dfs(s,inf))
        max_flow+=delta;
    }
    // todos los nodos con dist[u]!=-1 vs los que tienen
        dist[v]==-1 forman el min-cut, (u,v)
    return max_flow;
  }

  vector<ii> cut;
  vector<bool> seen;
  void dfs_cut(int u){
    seen[u] = 1;
    for(edge &e : g[u]){
      if(!seen[e.to]){
        if(dist[e.to]==-1 && e.cap!=0){
          cut.pb({u,e.to});
        }else if(e.f < e.cap){
          dfs_cut(e.to);
        }
      }
    }
  }
  vector<ii> min_cut(){
    seen.assign(n, false);
    dfs_cut(s);
    sort(all(cut));
    cut.resize(unique(all(cut)) - cut.begin());
    return cut;
  }
};
```

4.3 Push-Relabel

```
// Complexity O(V^2 * sqrt(E)) o O(V^3)
const ll inf = 1e17;
struct PushRelabel{
  struct edge {
    int to, rev; ll f, cap;
    edge(int to, int rev, ll cap, ll f = 0) : to(to), rev
        (rev), f(f), cap(cap) {}
  };
  void addEdge(int s, int t, ll cap){
    g[s].pb(edge(t, sz(g[t]), cap));
    g[t].pb(edge(s, sz(g[s])-1, (ll)0));
  }

  int n, s, t;
  vi height; vector<ll> excess;
  vector<vector<edge>> g;

  PushRelabel(int n_){
    n = n_; g.resize(n);
  }
  void push(int u, edge &e){
    ll d = min(excess[u], e.cap - e.f);
    edge &rev = g[e.to][e.rev];
    e.f += d;  rev.f -= d;
    excess[u] -= d;  excess[e.to] += d;
  }
  void relabel(int u){
    ll d = inf;
    for (edge e : g[u])
      if (e.cap - e.f > 0)
        d = min(d,(ll) height[e.to]);

    if (d < inf) height[u] = d + 1;
  }
  vi find_max_height_vertices(int s, int t) {
    vi max_height;
    for (int i = 0; i < n; i++)
      if (i != s && i != t && excess[i] > 0) {
        if (!max_height.empty() && height[i] > height[
            max_height[0]])
          max_height.clear();
        if (max_height.empty() || height[i] == height[
            max_height[0]])
          max_height.push_back(i);
      }
    return max_height;
  }

  ll maxFlow(){
```

```
        height.assign(n,0); excess.assign(n,0);
        ll max_flow = 0; bool pushed;
        vi current;

        height[s] = n; excess[s] = inf;
        for(edge &e: g[s])
            push(s,e);

        while(!(current = find_max_height_vertices(s,t)).
            empty()){
          for(int v: current){
            pushed = false;
            if(excess[v]==0) continue;
            for(edge &e : g[v]){
              if(e.cap - e.f>0 && height[v]== height[e.to]+1)
                  {
                pushed = true;
                push(v,e);
              }
            }
            if(!pushed){
              relabel(v);
              break;
            }
          }
        }
        for (edge e : g[t]){
          edge rev = g[e.to][e.rev];
          max_flow += rev.f;
        }
        return max_flow;
    }
};
```

4.4 Konig

```
#define sz(c) ((int)c.size())
// asume que el dinic YA ESTA tirado
// asume que nodes-1 y nodes-2 son la fuente y destino
int match[maxnodes]; // match[v]=u si u-v esta en el
    matching, -1 si v no esta matcheado
int s[maxnodes]; // numero de la bfs del koning
queue<int> kq;
// s[e]%2==1  o si e esta en V1 y s[e]==-1-> lo agarras
void konig() {//O(n)
        forn(v,nodes-2) s[v] = match[v] = -1;
        forn(v,nodes-2)
        for(edge it: g[v])
            if (it.to < nodes-2 && it.f>0){
                match[v]=it.to; match[it.to]=v;
            }
        forn(v,nodes-2)
        if (match[v]==-1){
            s[v]=0;kq.push(v);
        }
        while(!kq.empty()) {
                int e = kq.front(); kq.pop();
                if (s[e]%2==1) {
                        s[match[e]] = s[e]+1;
                        kq.push(match[e]);
                } else {
                        for(edge it: g[e])
                if (it.to < nodes-2 && s[it.to]==-1){
                    s[it->to] = s[e]+1;
                    kq.push(it->to);
                }
                }
        }
}
```

4.5 MCBM Augmenting Algorithm

```
// O (V*E)
vi g[nax], seen, match;
int Aug(int l) {                    // return 1 if an
    augmenting path is found
  if (seen[l]) return 0;       // return 0 otherwise
  seen[l] = 1;
  for (int r: g[l])
    if (match[r] == -1 || Aug(match[r])){
      match[r] = l; return 1;
        // found 1 matching
    }
  return 0;
                                                          //
      no matching
}

int MCBM(int n, int vleft){
  int ans = 0;
  match.assign(n, -1);     // V is the number of vertices
      in bipartite graph
  forn(l,vleft) {          // vleft : vi with indices of
      vertices
    seen.assign(n, 0);                      // reset before
        each recursion
    ans += Aug(l);
  }
  return ans;
}
```

4.6 Hungarian Algorithm

```
// Complexity O(V^3) maximiza
const int nax = 300;
const int inf = 1e9;
```

```
struct KM {
  int W[nax][nax], n;
        int mx[nax], my[nax]; // match arr
        int lx[nax], ly[nax]; // label arrMAXN
        int x[nax], y[nax]; // used arr
        int hungary(int nd) {
    int i;
    x[nd] = 1;
    forn(i,n)
      if(y[i] == 0 && W[nd][i] == lx[nd]+ly[i]) {
        y[i] = 1;
        if(my[i] == -1 || hungary(my[i])) {
          my[i] = nd;
          return 1;
        }
      }
    return 0;
        }
        int run() {
    int k, d;
    memset(mx, -1, sizeof(mx));
    memset(my, -1, sizeof(my));
    forn(i,n)
      lx[i] = 0, ly[i] = 0;
    forn(i,n)
      forn(j,n)
        lx[i] = max( lx[i] ,W[i][j]);
    forn(i,n) {
      while(1) {
        memset(x, 0, sizeof(x));
        memset(y, 0, sizeof(y));
        if(hungary(i))  break;
        d = inf;
        forn(j,n) {
          if(x[j]) {
            forn(k,n)
              if(!y[k])
                d = min(d,lx[j]+ly[k]-W[j][k]);
          }
        }
        if(d == inf)  break;
        forn(j,n) {
          if(x[j])    lx[j] -= d;
          if(y[j])    ly[j] += d;
        }
      }
    }
    int res = 0;
    forn(i,n) {
      if(my[i] != -1)
        res += W[my[i]][i];
    }
    return res;
        }
} km;
```

4.7 Min-Cost Max-Flow Algorithm

```
///Complexity O(V^2 * E^2)
const ll inf = 1e18;
struct edge {
  int to, rev; ll f, cap, cost;
  edge(int to, int rev, ll cap, ll cost, ll f=0) : to(to)
     , rev(rev), cap(cap), cost(cost), f(f) {}
};
struct MCMF{
  int n;
  vector<vector<edge>> g;
  void addEdge(int s, int t, ll cap, ll cost){
    g[s].pb(edge(t, sz(g[t]), cap, cost));
    g[t].pb(edge(s, sz(g[s])-1, 0, -cost));
  }
  MCMF(int n):n(n){
    g.resize(n);
  }
  void spfa(int v0, vector<ll>& d, vector<int>& p) {
    d.assign(n, inf); d[v0] = 0;
    vector<bool> inq(n,false);
    queue<int> q;
    q.push(v0);
    p.assign(n,-1);
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      inq[u] = false;
      for(int i= 0; i< g[u].size(); ++i){
        edge v = g[u][i];
        if (v.cap - v.f > 0 && d[v.to] > d[u] + v.cost )
            {
          d[v.to] = d[u] + v.cost;
          p[v.to] = v.rev;
          if (!inq[v.to]) {
            inq[v.to] = true;
            q.push(v.to);
          }
        }
      }
    }
  }
  ll min_cost_flow(ll K, int s, int t) {
    ll flow = 0, cost = 0;
    vector<int> p;
    vector<ll> d;
    while (flow < K) {
      spfa(s, d, p);
      if (d[t] == inf) break;
//        find max flow on that path
```

```
        ll f = K - flow;
        int cur = t;
        while (cur != s) {
          int u = g[cur][p[cur]].to;
          int rev = g[cur][p[cur]].rev;
          ll c = g[u][rev].cap - g[u][rev].f;
          f = min(f, c);
          cur = u;
        }
        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
          int rev = g[cur][p[cur]].rev;
          int u = g[cur][p[cur]].to;
          g[u][rev].f += f;
          g[cur][p[cur]].f -= f;
          cur = u;
        }
      }
      if(flow< K) return -1;
      return cost;
    }
};
```

4.8 Min-Cost Max-Flow Algorithm 2

```
typedef ll tf;
typedef ll tc;
const tf INFFLOW=1e9;
const tc INFCOST=1e9;
struct MCF{
        int n;
        vector<tc> prio, pot; vector<tf> curflow; vector<
            int> prevedge,prevnode;
        priority_queue<pair<tc, int>, vector<pair<tc, int
            >>, greater<pair<tc, int>>> q;
        struct edge{int to, rev; tf f, cap; tc cost;};
        vector<vector<edge>> g;
        MCF(int n):n(n),prio(n),curflow(n),prevedge(n),
            prevnode(n),pot(n),g(n){}
        void add_edge(int s, int t, tf cap, tc cost) {
                g[s].pb((edge){t,sz(g[t]),0,cap,cost});
                g[t].pb((edge){s,sz(g[s])-1,0,0,-cost});
        }
        pair<tf,tc> get_flow(int s, int t) {
                tf flow=0; tc flowcost=0;
                while(1){
                        q.push({0, s});
                        fill(all(prio),INFCOST);
                        prio[s]=0; curflow[s]=INFFLOW;
                        tc d; int u;
                        while(sz(q)){
                                tie(d,u)=q.top(); q.pop()
                                    ;
                                if(d!=prio[u]) continue;
                                forn(i,sz(g[u])) {
                                        edge &e=g[u][i];
                                        int v=e.to;
                                        if(e.cap<=e.f)
                                            continue;
                                        tc nprio=prio[u]+
                                            e.cost+pot[u]-
                                            pot[v];
                                        if(prio[v]>nprio)
                                             {
                                                prio[v]=
                                                    nprio;
                                                q.push({
                                                    nprio,
                                                     v});
                                                prevnode[
                                                    v]=u;
                                                    prevedge
                                                    [v]=i;
                                                curflow[v
                                                    ]=min(
                                                    curflow
                                                    [u], e
                                                    .cap-e
                                                    .f);
                                        }
                                }
                        }
                        if(prio[t]==INFCOST) break;
                        forn(i,n) pot[i]+=prio[i];
                        tf df=min(curflow[t], INFFLOW-
                            flow);
                        flow+=df;
                        for(int v=t; v!=s; v=prevnode[v])
                             {
                                edge &e=g[prevnode[v]][
                                    prevedge[v]];
                                e.f+=df; g[v][e.rev].f-=
                                    df;
                                flowcost+=df*e.cost;
                        }
                }
                return {flow,flowcost};
        }
};
```

4.9 Blossom

```
/// Complexity: O(|E||V|^2)
/// Tested: https://tinyurl.com/oe5rnpk
/// Max matching undirected graph
```

```
struct network {
  struct struct_edge { int v; struct_edge * n; };
  typedef struct_edge* edge;
  int n;
  struct_edge pool[MAXE]; ///2*n*n;
  edge top;
  vector<edge> adj;
  queue<int> q;
  vector<int> f, base, inq, inb, inp, match;
  vector<vector<int>> ed;
  network(int n) : n(n), match(n, -1), adj(n), top(pool),
      f(n), base(n),
                 inq(n), inb(n), inp(n), ed(n, vector<
                    int>(n)) {}
  void add_edge(int u, int v) {
    if(ed[u][v]) return;
    ed[u][v] = 1;
    top->v = v, top->n = adj[u], adj[u] = top++;
    top->v = u, top->n = adj[v], adj[v] = top++;
  }
  int get_lca(int root, int u, int v) {
    fill(inp.begin(), inp.end(), 0);
    while(1) {
      inp[u = base[u]] = 1;
      if(u == root) break;
      u = f[ match[u] ];
    }
    while(1) {
      if(inp[v = base[v]]) return v;
      else v = f[ match[v] ];
    }
  }
  void mark(int lca, int u) {
    while(base[u] != lca) {
      int v = match[u];
      inb[ base[u ]] = 1;
      inb[ base[v] ] = 1;
      u = f[v];
      if(base[u] != lca) f[u] = v;
    }
  }
  void blossom_contraction(int s, int u, int v) {
    int lca = get_lca(s, u, v);
    fill(all(inb), 0);
    mark(lca, u); mark(lca, v);
    if(base[u] != lca) f[u] = v;
    if(base[v] != lca) f[v] = u;
    forn(u,n){
      if(inb[base[u]]) {
        base[u] = lca;
        if(!inq[u]) {
            inq[u] = 1;
            q.push(u);
        }
      }
    }
  }
  int bfs(int s) {
    fill(all(inq), 0);
    fill(all(f), -1);
    for(int i = 0; i < n; i++) base[i] = i;
    q = queue<int>();
    q.push(s);
    inq[s] = 1;
   while(sz(q)) {
      int u = q.front(); q.pop();
      for(edge e = adj[u]; e; e = e->n) {
        int v = e->v;
        if(base[u] != base[v] && match[u] != v) {
          if((v == s) || (match[v] != -1 && f[match[v]]
              != -1))
            blossom_contraction(s, u, v);
          else if(f[v] == -1) {
            f[v] = u;
            if(match[v] == -1) return v;
            else if(!inq[match[v]]) {
              inq[match[v]] = 1;
              q.push(match[v]);
            }
          }
        }
      }
    }
    return -1;
  }
  int doit(int u) {
    if(u == -1) return 0;
    int v = f[u];
    doit(match[v]);
    match[v] = u; match[u] = v;
    return u != -1;
  }
  /// (i < net.match[i]) => means match
  int maximum_matching() {
    int ans = 0;
    forn(u,n)
      ans += (match[u] == -1) && doit(bfs(u));
    return ans;
  }
};
```

5 Data Structures

5.1 Disjoint Set Union

```
// Complejidad aprox O(1)
struct dsu{
```

```cpp
  vi p, r; int num_sets;
  void init(int n){
    p.assign(n, 0), r.assign(n, 1), num_sets = n;
    iota(all(p), 0);
  }
  int find_set(int i){
    return (p[i] == i ? i : p[i] = find_set(p[i]));
  }
  bool is_same_set(int i, int j){
    return find_set(i) == find_set(j);
  }
  void union_set(int i, int j){
    int x = find_set(i), y = find_set(j);
    if(x == y) return;
    if(r[x] > r[y]) swap(x, y);
    p[x] = y;
    r[y] += r[x], r[x] = 0;
    --num_sets;
  }
};
```

5.2 SQRT Decomposition

```cpp
// Complexity
// Preprocessing O(n)  query O(n/sqrt(n) + sqrt(n))
// Update O(1)
struct sqrt_decomp{
  vi a, b;
  int n, len;
  sqrt_decomp(vi &arr){ // preprocessing
    a = arr; n = sz(a); len = sqrt(n) + 1;
    b = vi(len);
    forn(i,n) b[i/len] += a[i];
  }
  void update(int pos, int val){
    int bpos = pos/len;
    b[bpos] += val - a[pos];
    a[pos] = val;
  }
  int query(int l, int r){
    int sum = 0;
    int c_l = l / len,   c_r = r / len;
    if (c_l == c_r){
      fore(i,l,r) sum += a[i];
    }else{
      fore(i,l,(c_l+1)*len-1) sum += a[i];
      fore(i,c_l+1,c_r-1) sum += b[i];
      fore(i,c_r*len,r) sum += a[i];
    }
    return sum;
  }
};
```

5.3 Fenwick Tree

```cpp
struct fwtree{ // 1-indexed
  vector<int> bit;
  int n;
  fwtree(int n):n(n){
    bit.assign(n + 1, 0);
  }
  int rsq(int r) {
    int sum = 0; for (; r; r -= r&-r ) sum += bit[r];
    return sum;
  }
  int rsq(int l, int r) {
    return rsq(r) - (l == 1 ? 0 : rsq(l - 1));
  }
  void upd(int r, int v) {
    for (; r <= n; r += r&-r) bit[r] += v;
  }
};
```

5.4 Fenwick Tree 2D

```cpp
struct fwtree{ /// 1-indexed
  vector<vector<ll>> bit;
  int n, m;
  fwtree(){}
  fwtree(int n, int m):n(n),m(m){
    bit = vector<vector<ll>>(n+1, vector<ll>(m+1,0));
  }
  ll sum(int x, int y) {
    ll v = 0;
    for (int i = x; i > 0; i -= i&(-i) )
      for (int j = y; j > 0; j -= j&(-j) )
        v += bit[i][j];
    return v;
  }

  void add(int x, int y, ll dt) {
    for (int i = x; i <= n; i += i&(-i) )
      for (int j = y; j <= m; j += j&(-j) )
        bit[i][j] += dt;
  }
};
```

5.5 Segment Tree

```cpp
#define neutro 0
struct stree{
  int n;  vector<int> t;
  stree(int m){
```

```
    n = m;   t.resize(n<<2);
  }
  stree(vector<int> &a){
    n = sz(a);
    t.resize(n<<2);
    build(1,0, n-1, a);
  }
  inline int oper(int a, int b){  return a+b; }
  void build(int v, int tl, int tr, vector<int> &a){
    if(tl==tr){
      t[v]= a[tl];  return ;
    }
    int tm = (tr+tl)/ 2;
    build(v*2, tl, tm, a);
    build(v*2+1, tm+1, tr, a);
    t[v] = oper(t[v*2],t[v*2+1]);
  }
  int query(int v, int tl, int tr, int l, int r){
    if(tl>r || tr<l) return neutro;
    if(l<=tl && tr<=r) return t[v];
    int tm = (tl+tr)/2;
    return oper(query(v*2, tl, tm, l, r),
           query(v*2+1, tm+1, tr, l, r));
  }
  void upd(int v, int tl, int tr, int pos, int val){
    if(tl==tr){
      t[v] = val;  return ;
    }
    int tm = (tr+tl)/ 2;
    if(pos<= tm)  upd(v*2, tl, tm, pos, val);
    else  upd(v*2+1, tm+1, tr, pos, val);
    t[v] = oper(t[v*2], t[v*2+1]);
  }
  void upd(int pos, int val){ upd(1,0,n-1,pos,val); }
  int query(int l, int r){ return query(1,0,n-1,l,r); }
};
```

5.6 ST Lazy Propagation

```
#define neutro -1e9
struct stree{
  int n;   vector<int> t, lazy;
  stree(int m){
    n = m;    t.resize(n<<2);
    lazy.resize(n<<2);
  }
  stree(vector<int> &a){
    n = sz(a); t.resize(n<<2);   lazy.resize(n<<2);
    build(1,0, n-1, a);
  }
  inline int oper(int a, int b){ return max(a,b); }
  void build(int v, int tl, int tr, vector<int> &a){
    if(tl==tr){
      t[v]= a[tl]; return ;
    }
    int tm = tl + (tr-tl)/ 2;
    build(v*2, tl, tm, a);
    build(v*2+1, tm+1, tr, a);
    t[v] = oper(t[v*2], t[v*2+1]);
  }
  void push(int v) {
    t[v*2] += lazy[v]; lazy[v*2] += lazy[v];
    t[v*2+1] += lazy[v]; lazy[v*2+1] += lazy[v];
    lazy[v] = 0;
  }
  void upd(int v, int tl, int tr, int l, int r, int val)
     {
    if(tl>r || tr<l) return ;
    if (l <= tl && tr <= r) {
      t[v] += val;
      lazy[v] += val; return ;
    }
    push(v);
    int tm = tl + (tr-tl)/ 2;
    upd(v*2, tl, tm, l, r, val);
    upd(v*2+1, tm+1, tr, l, r, val);
    t[v] = oper(t[v*2], t[v*2+1]);
  }
  int query(int v, int tl, int tr, int l, int r) {
    if(tl>r || tr<l) return neutro;
    if (l <= tl && tr <= r) return t[v];
    push(v);
    int tm = tl + (tr-tl)/ 2;
    return oper(query(v*2, tl, tm, l, r),
           query(v*2+1, tm+1, tr, l, r));
  }
  void upd(int l, int r, int val){ upd(1,0,n-1,l, r,val);
     }
  int query(int l, int r){ return query(1,0,n-1,l,r); }
};
```

5.7 Persistent ST

```
#define neutro 0
struct node{
  int sum, l, r;
};
struct stree{
  vector<int> rts;
  vector<node> t;
  int n, idx;
  inline int oper(int a, int b){ return a+b; }
  int build(int tl, int tr, vector<int> &a){
    int v = idx++;
    if(tl==tr){
      t[v].sum = a[tl]; return v;
```

```
    }
    int tm = (tl+tr)/2;
    t[v].l = build(tl, tm, a);
    t[v].r = build(tm+1, tr, a);
    t[v].sum = t[t[v].l].sum + t[t[v].r].sum;
    return v;
  }
  stree(vector<int> &a){
    n = sz(a);
    t.resize(# define nax);
    idx = 0;
    rts.pb(0);
    build(0, n-1, a);
  }
  int query(int v, int tl, int tr, int l, int r){
    if(tl>r || tr<l) return neutro;
    if(l<=tl && tr<= r){
      return t[v].sum;
    }
    int tm = (tl+tr)/2;
    return oper(query(t[v].l, tl, tm, l, r), query(t[v].r
       , tm+1, tr, l, r));
  }
  int upd(int prev, int tl, int tr, int pos, int val){
    int v = idx++; t[v] = t[prev];
    if(tl==tr){
      t[v].sum = val; return v;
    }
    int tm = (tl+tr)/2;
    if(pos<=tm) t[v].l = upd(t[v].l, tl, tm, pos, val);
    else t[v].r = upd(t[v].r, tm+1, tr, pos, val);

    t[v].sum = t[t[v].l].sum + t[t[v].r].sum;
    return v;
  }
  int query(int v, int l, int r){
    return query(v, 0, n-1, l, r);
  }

  void upd(int pos, int val){
    int id = upd(rts.back(), 0, n-1, pos, val);
    rts.pb(id);
  }
};
```

5.8 Segtree 2D

```
int n,m;
const ll neutro = 0;
ll op(ll a, ll b){ return a+b;}
ll a[nax][nax],st[2*nax][2*nax];
void build(){
        forn(i,n)forn(j,m)st[i+n][j+m]=a[i][j];
        forn(i,n)for(int j=m-1;j;--j)
                st[i+n][j]=op(st[i+n][j<<1],st[i+n][j
                   <<1|1]);
        for(int i=n-1;i;--i)forn(j,2*m)
                st[i][j]=op(st[i<<1][j],st[i<<1|1][j]);
}
void upd(int x, int y, ll v){
        st[x+n][y+m]=v;
        for(int j=y+m;j>1;j>>=1)st[x+n][j>>1]=op(st[x+n][
            j],st[x+n][j^1]);
        for(int i=x+n;i>1;i>>=1)for(int j=y+m;j;j>>=1)
                st[i>>1][j]=op(st[i][j],st[i^1][j]);
}
ll query(int x0, int x1, int y0, int y1){ // [x0, x1) , [
   y0,y1)
        ll r=neutro;
        for(int i0=x0+n,i1=x1+n;i0<i1;i0>>=1,i1>>=1){
                int t[4],q=0;
                if(i0&1)t[q++]=i0++;
                if(i1&1)t[q++]=--i1;
                forn(k,q)for(int j0=y0+m,j1=y1+m;j0<j1;j0
                   >>=1,j1>>=1){
                        if(j0&1)r=op(r,st[t[k]][j0++]);
                        if(j1&1)r=op(r,st[t[k]][--j1]);
                }
        }
        return r;
}
```

5.9 RSQ

```
//K  has to satisfy K> log nax + 1
ll st[nax][K], a[nax], sum = 0;
void init(int N){
  forn(i,N) st[i][0] = a[i];
  for1(j,K-1)
    forn(i,N-(1 << j)+1)
      st[i][j] = st[i][j-1] + st[i + (1 << (j - 1))][j -
         1];
}
void get(int l, int r){
  sum = 0;
  for (int j = K-1; j >= 0; j--) {
    if ((1 << j) <= r - l + 1) {
      sum += st[l][j];
      l += 1 << j;
    }
  }
}
```

5.10 RMQ

```
//K  has to satisfy K> log nax + 1
```

```
int st[nax][K], a[nax];
int logp[nax];
void init(int N){
//  logp[1] = 0;
//  for (int i = 2; i < nax; i++) logp[i] = logp[i/2] +
    1;

  forn(i,N) st[i][0] = a[i];
  for1(j,K-1)
    forn(i,N-(1 << j)+1)
      st[i][j] = min(st[i][j-1], st[i + (1 << (j - 1))][j
        - 1]);
}

int get(int l, int r){ //assuming L<=R
  if(l>r) return inf;
//  int j = logp[r - l + 1];
  int j = 31 - __builtin_clz(r-l+1);
  return min(st[l][j], st[r - (1 << j) + 1][j]);
}
```

5.11 Sack

```
// Time Complexity O(N*log(N))
int timer;
int cnt[nax], big[nax], fr[nax], to[nax], who[nax];
vector<int> g[nax];
int pre(int u, int p){
  int sz = 1, tmp;
  who[timer] = u;
  fr[u] = timer++;
  ii best = {-1, -1};
  for(int v: g[u]){
    if(v==p) continue;
    tmp = pre(v,u);
    sz+=tmp;
    best = max(best, {tmp, v});
  }
  big[u] = best.se;
  to[u] = timer-1;
  return sz;
}
void add(int u, int x) { /// x == 1 add, x == -1 delete
  cnt[u] += x;
}
void dfs(int u, int p, bool keep = true){
  for(int v: g[u])
    if(v!=p && v!=big[u])
      dfs(v,u, 0);
  if(big[u]!=-1) dfs(big[u], u);
  /// add all small
  for(int v: g[u])
    if(v!=p && v!=big[u])
      for(int i = fr[v]; i<= to[v]; ++i)
        add(who[i],1);
  add(u,1);
  /// Answer queries
  if(!keep)
    for(int i = fr[u]; i<= to[u]; ++i)
      add(who[i],-1);
}
void solve(int root){
  timer = 0;
  pre(root, root);
  dfs(root, root);
}
```

5.12 Heavy Light Decomposition

```
vector<int> g[nax];
int len[nax], dep[nax], in[nax], out[nax], head[nax], par
    [nax], idx;
void dfs_sz( int u, int d ) {
  dep[u] = d;
  int &sz = len[u]; sz = 1;
  for( auto &v : g[u] ) {
    if( v == par[u] ) continue;
    par[v] = u; dfs_sz(v, d+1);
    sz += len[v];
    if(len[ g[u][0] ] < len[v]) swap(g[u][0], v);
  }
  return ;
}
void dfs_hld( int u) {
  in[u] = idx++;
  arr[in[u]] = val[u]; /// to initialize the segment tree
  for( auto& v : g[u] ) {
    if( v == par[u] ) continue;
    head[v] = (v == g[u][0] ? head[u] : v);
    dfs_hld(v);
  }
  out[u] = idx-1;
}
void upd_hld( int u, int val ) {
  upd_DS(in[u], val);
}
int query_hld( int u, int v ) {
  int val = neutro;
  while( head[u] != head[v] ) {
    if( dep[ head[u] ] < dep[ head[v] ] ) swap(u, v);
    val = val + query_DS(in[ head[u] ], in[u]);
    u = par[ head[u] ];
  }
  if( dep[u] > dep[v] ) swap(u, v);
  val = val+query_DS(in[u], in[v]);
  return val;
/// when updates are on edges use: (line 36)
```

```
///   if (dep[u] == dep[v]) return val;
///   val = val+query_DS(in[u] + 1, in[v]);
}
void build(int root) {
  idx = 0; /// DS index [0, n)
  par[root] = head[root] = root;
  dfs_sz(root, 0);
  dfs_hld(root);
  /// initialize DS
}
```

5.13 Treap

```
typedef struct item *pitem;
struct item {
	int pr,key,cnt;
	pitem l,r;
	item(int key):key(key),pr(rand()),cnt(1),l(0),r
	    (0) {}
};
int cnt(pitem t){return t?t->cnt:0;}
void upd_cnt(pitem t){if(t)t->cnt=cnt(t->l)+cnt(t->r)+1;}
void split(pitem t, int key, pitem& l, pitem& r){ // l: <
    key, r: >= key
	if(!t)l=r=0;
	else if(key<t->key)split(t->l,key,l,t->l),r=t;
	else split(t->r,key,t->r,r),l=t;
	upd_cnt(t);
}
void insert(pitem& t, pitem it){
	if(!t)t=it;
	else if(it->pr>t->pr)split(t,it->key,it->l,it->r)
	    ,t=it;
	else insert(it->key<t->key?t->l:t->r,it);
	upd_cnt(t);
}
void merge(pitem& t, pitem l, pitem r){
	if(!l||!r)t=l?l:r;
	else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
	else merge(r->l,l,r->l),t=r;
	upd_cnt(t);
}
void erase(pitem& t, int key){
	if(t->key==key)merge(t,t->l,t->r);
	else erase(key<t->key?t->l:t->r,key);
	upd_cnt(t);
}
void unite(pitem &t, pitem l, pitem r){
	if(!l||!r){t=l?l:r;return;}
	if(l->pr<r->pr)swap(l,r);
	pitem p1,p2;split(r,l->key,p1,p2);
	unite(l->l,l->l,p1);unite(l->r,l->r,p2);
	t=l;upd_cnt(t);
}
pitem kth(pitem t, int k){
	if(!t)return 0;
	if(k==cnt(t->l))return t;
	return k<cnt(t->l)?kth(t->l,k):kth(t->r,k-cnt(t->
	    l)-1);
}
pair<int,int> lb(pitem t, int key){ // position and value
     of lower_bound
	if(!t)return {0,1<<30}; // (special value)
	if(key>t->key){
		auto w=lb(t->r,key);w.fst+=cnt(t->l)+1;
		    return w;
	}
	auto w=lb(t->l,key);
	if(w.fst==cnt(t->l))w.snd=t->key;
	return w;
}
```

5.14 Implicit Treap

```
// example that supports range reverse and addition
    updates, and range sum query
// (commented parts are specific to this  problem)
typedef struct item* pitem;
struct item {
	int pr,cnt,val;
//	int sum; // (paramters for range query)
//	bool rev;int add; // (parameters for lazy prop)
	pitem l,r;
	item(int val): pr(rand()),cnt(1),val(val),l(0),r
	    (0)/*,sum(val),rev(0),add(0)*/ {}
};
void push(pitem it){
	if(it){
		/*if(it->rev){
			swap(it->l,it->r);
			if(it->l)it->l->rev^=true;
			if(it->r)it->r->rev^=true;
			it->rev=false;
		}
		it->val+=it->add;it->sum+=it->cnt*it->add
		    ;
		if(it->l)it->l->add+=it->add;
		if(it->r)it->r->add+=it->add;
		it->add=0;*/
	}
}
int cnt(pitem t){return t?t->cnt:0;}
// int sum(pitem t){return t?push(t),t->sum:0;}
void upd_cnt(pitem t){
	if(t){
		t->cnt=cnt(t->l)+cnt(t->r)+1;
		// t->sum=t->val+sum(t->l)+sum(t->r);
```

```
		}
	}
	void merge(pitem& t, pitem l, pitem r){
		push(l);push(r);
		if(!l||!r)t=l?l:r;
		else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
		else merge(r->l,l,r->l),t=r;
		upd_cnt(t);
	}
	void split(pitem t, pitem& l, pitem& r, int sz){ // sz: desired size of l
		if(!t){l=r=0;return;}
		push(t);
		if(sz<=cnt(t->l))split(t->l,l,t->l,sz),r=t;
		else split(t->r,t->r,r,sz-1-cnt(t->l)),l=t;
		upd_cnt(t);
	}
	void output(pitem t){ // useful for debugging
		if(!t)return;
		push(t);
		output(t->l);printf(" %d",t->val);output(t->r);
	}
	// use merge and split for range updates and queries
```

5.15 Implicit Treap Father

```
	// node father is useful to keep track of the chain of each node
	// alternative: splay tree
	// IMPORTANT: add pointer f in struct item
	void merge(pitem& t, pitem l, pitem r){
		push(l);push(r);
		if(!l||!r)t=l?l:r;
		else if(l->pr>r->pr)merge(l->r,l->r,r),l->r->f=t=l;
		else merge(r->l,l,r->l),r->l->f=t=r;
		upd_cnt(t);
	}
	void split(pitem t, pitem& l, pitem& r, int sz){
		if(!t){l=r=0;return;}
		push(t);
		if(sz<=cnt(t->l)){
			split(t->l,l,t->l,sz);r=t;
			if(l)l->f=0;
			if(t->l)t->l->f=t;
		}
		else {
			split(t->r,t->r,r,sz-1-cnt(t->l));l=t;
			if(r)r->f=0;
			if(t->r)t->r->f=t;
		}
		upd_cnt(t);
	}
	void push_all(pitem t){
		if(t->f)push_all(t->f);
		push(t);
	}
	pitem root(pitem t, int& pos){ // get root and position for node t
		push_all(t);
		pos=cnt(t->l);
		while(t->f){
			pitem f=t->f;
			if(t==f->r)pos+=cnt(f->l)+1;
			t=f;
		}
		return t;
	}
```

5.16 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
//methods
tree.find_by_order(k) //returns pointer to the k-th smallest element
tree.order_of_key(x)  //returns how many elements are smaller than x
//if element does not exist
tree.end() == tree.find_by_order(k) //true
```

5.17 Mo's Algorithm

```
/// Complexity: O(|N+Q|*sqrt(|N|)*|ADD/DEL|)
/// Requires add(), delete() and get_ans()
struct query {
  int l, r, idx;
};
int S; // s = sqrt(n)
bool cmp (query a, query b) {
  int x = a.l/S;
  if (x != b.l/S) return x < b.l/S;
  return (x&1 ? a.r < b.r : a.r > b.r);
}
void solve(){
  S = sqrt(n); // n = size of array
  sort(all(q), cmp);
  int l = 0, r = -1;
  forn(i, sz(q)){
    while (r < q[i].r) add(++r);
    while (l > q[i].l) add(--l);
    while (r > q[i].r) del(r--);
```

```
      while (l < q[i].l) del(l++);
      ans[q[i].idx] = get_ans();
    }
  }
```

6 Math

6.1 Sieve of Eratosthenes

```
// O(n)
// pr contains prime numbers
// lp[i] == i if i is prime
// else lp[i] is minimum prime factor of i
const int nax = 1e7;
int lp[nax+1];
vector<int> pr; // It can be sped up if change for an
   array

void sieve(){
  fore(i,2,nax-1){
    if (lp[i] == 0) {
      lp[i] = i; pr.pb(i);
    }
    for (int j=0, mult= i*pr[j]; j<sz(pr) && pr[j]<=lp[i]
        && mult<nax; ++j, mult= i*pr[j])
      lp[mult] = pr[j];
  }
}
```

6.2 Count primes

```
int count_primes(int n) {
  const int S = 10000;
  vector<int> primes;
  int nsqrt = sqrt(n);
  vector<char> is_prime(nsqrt + 1, true);
  fore(i,2,nsqrt){
    if (is_prime[i]) {
      primes.pb(i);
      for (int j = i * i; j <= nsqrt; j += i)
        is_prime[j] = false;
    }
  }

  int result = 0;
  vector<char> block(S);
  for (int k = 0; k * S <= n; k++) {
    fill(all(block), true);
    int start = k * S;
    for (int p : primes) {
      int start_idx = (start + p - 1) / p;
      int j = max(start_idx, p) * p - start;
      for (; j < S; j += p)
        block[j] = false;
    }
    if (k == 0)
      block[0] = block[1] = false;
    for (int i = 0; i < S && start + i <= n; i++) {
      if (block[i])
        result++;
    }
  }
  return result;
}
```

6.3 Segmented Sieve

```
// Complexity O((R-L+1)*log(log(R)) + sqrt(R)*log(log(R))
   )
// R-L+1 roughly 1e7  R-- 1e12
vector<bool> segmentedSieve(ll L, ll R) {
  // generate all primes up to sqrt(R)
  ll lim = sqrt(R);
  vector<bool> mark(lim + 1, false);
  vector<ll> primes;
  for (ll i = 2; i <= lim; ++i) {
    if (!mark[i]) {
      primes.emplace_back(i);
      for (ll j = i * i; j <= lim; j += i)
        mark[j] = true;
    }
  }
  vector<bool> isPrime(R - L + 1, true);
  for (ll i : primes)
    for (ll j = max(i * i, (L + i - 1) / i * i); j <= R;
       j += i)
      isPrime[j - L] = false;
  if (L == 1)
    isPrime[0] = false;
  return isPrime;
}
```

6.4 Polynomial Multiplication

```
int ans[grado1+grado2+1];

forn(c,grado1+grado2+1) ans[c] = 0;
forn(pos,grado1+1){
  forn(ter,grado2+1)
    ans[pos + ter] += pol1[pos] * pol2[ter];
}
```

6.5 Fast Fourier Transform

```
typedef double ld;
const ld PI = acos(-1.0L);
const ld one = 1;

typedef complex<ld> C;
typedef vector<ld> vd;

void fft(vector<C>& a) {
	int n = sz(a), L = 31 - __builtin_clz(n);
	static vector<complex<ld>> R(2, 1);
	static vector<C> rt(2, 1);  // (^ 10% faster if double)
	for (static int k = 2; k < n; k *= 2) {
		R.resize(n); rt.resize(n);
		auto x = polar(one, PI / k);
		fore(i,k,2*k-1) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
	}
	vi rev(n);
	forn(i,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
	forn(i,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
	for (int k = 1; k < n; k *= 2)
		for (int i = 0; i < n; i += 2 * k) forn(j,k) {
			// C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)  /// include-line
			auto x = (ld *)&rt[j+k], y = (ld *)&a[i+j+k];        /// exclude-line
			C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] + x[1]*y[0]);           /// exclude-line
			a[i + j + k] = a[i + j] - z;
			a[i + j] += z;
		}
}

typedef vector<ll> vl;

vl conv(const vl& a, const vl& b) {
	if (a.empty() || b.empty()) return {};
	vl res(sz(a) + sz(b) - 1);
	int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
	vector<C> in(n), out(n);
	copy(all(a), begin(in));
	forn(i,sz(b)) in[i].imag(b[i]);
	fft(in);
	for (C& x : in) x *= x;
	forn(i,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
	fft(out);
	forn(i,sz(res)) res[i] = floor(imag(out[i]) / (4 * n) +0.5);
	return res;
}

vl convMod(const vl &a, const vl &b, const int &M) {
	if (a.empty() || b.empty()) return {};
	vl res(sz(a) + sz(b) - 1);
	int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
	vector<C> L(n), R(n), outs(n), outl(n);
	forn(i,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
	forn(i,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
	fft(L), fft(R);
	forn(i,n) {
		int j = -i & (n - 1);
		outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
		outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
	}
	fft(outl), fft(outs);
	forn(i,sz(res)) {
		ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
		ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
		res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
	}
	return res;
}
```

6.6 FHT

```
ll c1[MAXN+9],c2[MAXN+9];  // MAXN must be power of 2 !!
void fht(ll* p, int n, bool inv){
	for(int l=1;2*l<=n;l*=2)for(int i=0;i<n;i+=2*l)forn(j,l){
		ll u=p[i+j],v=p[i+l+j];
		if(!inv)p[i+j]=u+v,p[i+l+j]=u-v; // XOR
		else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
		//if(!inv)p[i+j]=v,p[i+l+j]=u+v; // AND
		//else p[i+j]=-u+v,p[i+l+j]=u;
		//if(!inv)p[i+j]=u+v,p[i+l+j]=u; // OR
		//else p[i+j]=v,p[i+l+j]=u-v;
	}
}
// like polynomial multiplication, but XORing exponents
// instead of adding them (also ANDing, ORing)
vector<ll> multiply(vector<ll>& p1, vector<ll>& p2){
```

```
        int n=1<<(32-__builtin_clz(max(sz(p1),sz(p2))-1))
            ;
        forn(i,n)c1[i]=0,c2[i]=0;
        forn(i,sz(p1))c1[i]=p1[i];
        forn(i,sz(p2))c2[i]=p2[i];
        fht(c1,n,false);fht(c2,n,false);
        forn(i,n)c1[i]*=c2[i];
        fht(c1,n,true);
        return vector<ll>(c1,c1+n);
}
```

6.7 Fibonacci Matrix

```
// Pair Fn and Fn+1
ii fib (int n) {
  if (n == 0) return {0, 1};

  ii p = fib(n >> 1);
  int c = p.fi * (2 * p.se - p.fi);
  int d = p.fi * p.fi + p.se * p.se;
  if (n & 1) return ii(d, c + d);
  else return ii(c, d);
}
```

6.8 Matrix Exponentiation

```
struct matrix{ // define N
  int r, c, m[N][N];
  matrix(int r, int c):r(r),c(c){
    memset(m, 0, sizeof m);
  }
  matrix operator *(const matrix &b){
    matrix c = matrix(this->r, b.c);
    forn(i,this->r){
      forn(k,b.r){
        if(!m[i][k]) continue;
        forn(j,b.c){
          c.m[i][j] += m[i][k]*b.m[k][j];
        }
      }
    }
    return c;
  }
};
matrix pow(matrix &b, ll e){
  matrix c = matrix(b.r, b.c);
  forn(i,b.r) c.m[i][i] = 1;
  while(e){
    if(e&1LL) c = c*b;
    b = b*b , e/=2;
  }
  return c;
}
```

6.9 Binary Exponentiation

```
int binpow(int b, int e) {
        int ans = 1;
        for (; e; b = 1LL*b*b%mod, e /= 2)
                if (e&1) ans = 1LL*ans*b%mod;
        return ans;
}
```

6.10 Euler's Totient Function

```
int phi(int n) { // O(sqrt(n))
  if(n==1) return 0;
  int ans = n;
  for (int i = 2; 1ll*i*i <= n; i++) {
    if(n % i == 0) {
      while(n % i == 0) n /= i;
      ans -= ans / i;
    }
  }
  if(n > 1) ans -= ans / n;
  return ans;
}
//////////////////
vi phi_(int n) { // O(n loglogn)
  vi phi(n + 1);
  phi[0] = 0;
  for1(i,n) phi[i] = i;
  fore(i,2,n){
    if(phi[i] != i) continue;
    for (int j = i; j <= n; j += i)
      phi[j] -= phi[j] / i;
  }
}

////////// with linear sieve when i is not a prime number
if (lp[i] == lp[i / lp[i]])
  phi[i] = phi[i / lp[i]] * lp[i];
else
  phi[i] = phi[i / lp[i]] * (lp[i] - 1);
```

6.11 Extended Euclidean (Diophantic)

```
// a*x+b*y = g
ll gcde(ll a, ll b, ll& x, ll& y) {
  x = 1, y = 0;
  ll x1 = 0, y1 = 1, a1 = a, b1 = b;
  ll q;
  while (b1) {
    q = a1 / b1;
```

```
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}

bool find_any_solution(ll a, ll b, ll c, ll &x0, ll &y0,
    ll &g) {
  g = gcde(abs(a), abs(b), x0, y0);
  if (c % g) return false;

  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
```

6.12 Inversa modular

```
// O(mod)
const int mod;
int inv[mod];
void precalc(){
  inv[1] = 1;
  fore(i,2,mod-1) inv[i] = (mod - (mod/i) * inv[mod%i] %
      mod) % mod;
}

//////////////////////////////////////
ll inverse(ll a, ll m){
  ll x, y;
  ll g = gcde(a, m, x, y);
  if (g != 1) {
    cout << "No solution!";
    return -1;
  }else{
    x = (x % m + m) % m;
    return x;
  }
}
```

6.13 Legendre's Formula

```
// Complexity O(log_k (n))
// If k is prime
int fact_pow (int n, int k) {
  int x = 0;
  while(n) {
    n /= k; x += n;
  }
  return x;
}
// If k is composite k = k1^p1 * k2^p2 * ... * km^pm
// min 1..m   ai/ pi  where ai is fact_pow(n, ki)
```

6.14 Mobious

```
int mu[nax], f[nax], h[nax];
void pre(){
  mu[0] = 0; mu[1] = 1;
  for(int i = 1; i<nax; ++i){
    if(mu[i]==0) continue;
    for(int j= i+i; j<nax; j+=i){
      mu[j] -= mu[i];
    }
  }

  for(int i = 1; i < nax; ++i){
    for(int j = i; j < nax; j += i){
      f[j] += h[i]*mu[j/i];
    }
  }
}
///////
void pre(){
  mu[0] = 0; mu[1] = 1;
  fore(i,2,N){
    if (lp[i] == 0) {
      lp[i] = i; mu[i] = -1;
      pr.pb(i);
    }
    for (int j=0, mult= i*pr[j]; j<sz(pr) && pr[j]<=lp[i]
        && mult<=N; ++j, mult= i*pr[j]){
      if(i%pr[j]==0) mu[mult] = 0;
      else mu[mult] = mu[i]*mu[pr[j]];
      lp[mult] = pr[j];
    }
  }
}
```

6.15 Miller Rabin Test

```
ll mulmod(ll a, ll b, ll m) {
        ll r=a*b-(ll)((long double)a*b/m+.5)*m;
        return r<0?r+m:r;
}
ll binpow(ll b, ll e, ll m){
  ll r = 1;
  while(e){
    if(e&1) r = mulmod(r, b,m);
    b = mulmod(b,b,m);
    e = e/2;
  }
```

```
  return r;
}

bool is_prime(ll n, int a, ll s, ll d){
        if(n==a) return true;
        ll x=binpow(a,d,n);
        if(x==1 || x+1==n)return true;
        forn(k,s-1){
                x=mulmod(x,x,n);
                if(x==1) return false;
                if(x+1==n) return true;
        }
        return false;
}
int ar[]={2,3,5,7,11,13,17,19,23,29,31,37};
bool rabin(ll n){ // true iff n is prime
        if(n==2) return true;
        if(n<2 || n%2==0) return false;
        ll s=0,d=n-1;
        while(d%2==0)++s,d/=2;
        forn(i,12) if(!is_prime(n,ar[i],s,d)) return
            false;
  return true;
}

////////////////////////////////////////

bool isPrime(ll n) {
        if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
        ll A[] = {2, 325, 9375, 28178, 450775, 9780504,
            1795265022};
        ll s=0,d=n-1;
        while(d%2==0)++s,d/=2;
        for (ll a : A) {   // ^ count trailing zeroes
                ll p = binpow(a%n, d, n), i = s;
                while (p != 1 && p != n - 1 && a % n && i
                    --)
                        p = mulmod(p, p, n);
                if (p != n-1 && i != s) return 0;
        }
        return 1;
}
```

6.16 Pollard Rho

```
ll rho(ll n){
        if(!(n&1))return 2;
        ll x=2,y=2,d=1;
        ll c=rand()%n+1;
        while(d==1){
                x=(mulmod(x,x,n)+c)%n;
                y=(mulmod(y,y,n)+c)%n;
                y=(mulmod(y,y,n)+c)%n;
                if(x>=y)d=__gcd(x-y,n);
                else d=__gcd(y-x,n);
        }
        return d==n?rho(n):d;
}
void fact(ll n, map<ll,int>& f){ //O (lg n)^3
        if(n==1)return;
        if(rabin(n)){f[n]++;return;}
        ll q=rho(n);fact(q,f);fact(n/q,f);
}
```

6.17 Chinese Remainder Theorem

```
pll extendedEuclid(ll a, ll b){ // a * x + b * y = __gcd(
    a,b)
        ll x,y;
        if (b==0) return {1,0};
        auto p=extendedEuclid(b,a%b);
        x=p.se;
        y=p.fi-(a/b)*x;
        if(a*x+b*y==-__gcd(a,b)) x=-x, y=-y;
        return {x,y};
}
pair<pll,pll> diophantine(ll a, ll b, ll r) {
  //a*x+b*y=r where r is multiple of __gcd(a,b);
        ll d=__gcd(a,b);
  a/=d; b/=d; r/=d;
        auto p = extendedEuclid(a,b);
        p.fi*=r; p.se*=r;
        // assert(a*p.fi+b*p.se==r);
        return {p,{-b,a}}; // solutions: p+t*ans.se
}
ll inv(ll a, ll m) {
        assert(__gcd(a,m)==1);
        ll x = diophantine(a,m,1).fi.fi;
        return ((x%m)+m)%m;
}
#define MOD(a,m) (((a)%m+m)%m)
pll sol(tuple<ll,ll,ll> c){ //requires inv, diophantine
  ll a=get<0>(c), x1=get<1>(c), m=get<2>(c), d=__gcd(a,m)
    ;
  if(d==1) return pll(MOD(x1*inv(a,m),m), m);
  else return x1%d ? pll({-1LL,-1LL}) : sol(make_tuple(a/
      d,x1/d,m/d));
}
pair<ll,ll> crt(vector< tuple<ll,ll,ll> > &cond) { //
    returns: (sol, lcm)
        ll x1=0,m1=1,x2,m2;
        for(auto &t: cond){
                tie(x2,m2)=sol(t);
                if((x1-x2)%__gcd(m1,m2))return {-1,-1};
                if(m1==m2)continue;
                ll k=diophantine(m2,-m1,x1-x2).fi.se,l=m1
                    *(m2/__gcd(m1,m2));
                x1=MOD((__int128_t)m1*k+x1,l); m1=l;
```

```
        }
        return sol(make_tuple(1,x1,m1));
} //cond[i]={ai,bi,mi} ai*xi=bi (mi); assumes lcm fits in
    ll
```

6.18 Simplex

```
vector<int> X,Y;
vector<vector<double> > A;
vector<double> b,c;
double z;
int n,m;
void pivot(int x,int y){
        swap(X[y],Y[x]);
        b[x]/=A[x][y];
        forn(i,m)if(i!=y)A[x][i]/=A[x][y];
        A[x][y]=1/A[x][y];
        forn(i,n)if(i!=x&&abs(A[i][y])>eps){
                b[i]-=A[i][y]*b[x];
                forn(j,m)if(j!=y)A[i][j]-=A[i][y]*A[x][j
                    ];
                A[i][y]=-A[i][y]*A[x][y];
        }
        z+=c[y]*b[x];
        forn(i,m)if(i!=y)c[i]-=c[y]*A[x][i];
        c[y]=-c[y]*A[x][y];
}
pair<double,vector<double> > simplex( // maximize c^T x s
    .t. Ax<=b, x>=0
                vector<vector<double> > _A, vector<double
                    > _b, vector<double> _c){
        // returns pair (maximum value, solution vector)
        A=_A;b=_b;c=_c;
        n=sz(b);m=sz(c);z=0.;
        X=vector<int>(m);Y=vector<int>(n);
        forn(i,m)X[i]=i;
        forn(i,n)Y[i]=i+m;
        while(1){
                int x=-1,y=-1;
                double mn=-eps;
                forn(i,n)if(b[i]<mn)mn=b[i],x=i;
                if(x<0)break;
                forn(i,m)if(A[x][i]<-eps){y=i;break;}
                assert(y>=0); // no solution to Ax<=b
                pivot(x,y);
        }
        while(1){
                double mx=eps;
                int x=-1,y=-1;
                forn(i,m)if(c[i]>mx)mx=c[i],y=i;
                if(y<0)break;
                double mn=1e200;
                forn(i,n)if(A[i][y]>eps&&b[i]/A[i][y]<mn)
                        mn=b[i]/A[i][y],x=i;
                assert(x>=0); // c^T x is unbounded
                pivot(x,y);
        }
        vector<double> r(m);
        forn(i,n)if(Y[i]<m)r[Y[i]]=b[i];
        return {z,r};
}
```

6.19 Gauss Jordan

```
int gauss(vector<vector<double>> &a, vector<double> &ans)
    {
  int n = sz(a), m = sz(a[0]) - 1;
  vi where(m, -1);
  for(int col=0, row=0; col<m && row<n; ++col) {
    int sel = row;
    fore(i,row,n-1)
      if(abs(a[i][col]) > abs(a[sel][col])) sel = i;
    if(abs(a[sel][col]) < eps) continue;

    fore(i,col,m) swap (a[sel][i], a[row][i]);
    where[col] = row;

    forn(i,n){
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j) a[i][j] -= a[row][j] *
            c;
      }
    }
    ++row;
  }

  ans.assign(m, 0);
  forn(i,m){
    if(where[i] != -1) ans[i] = a[where[i]][m] / a[where[
        i]][i];
  }
  forn(i,n){
    double sum = 0;
    forn(j,m) sum += ans[j] * a[i][j];
    if(abs(sum - a[i][m]) > eps) return 0;
  }

  forn(i,m) if(where[i] == -1) return 1e9; /// infinitas
      soluciones
  return 1;
}
```

6.20 Gauss Jordan Modular

```
const int eps = 0, mod = 1e9+7;

int gauss(vector<vi> &a, vi &ans) {
```

```
    int n = sz(a), m = sz(a[0]) - 1;
    vi where(m, -1);
    for(int col=0, row=0; col<m && row<n; ++col) {
      int sel = row;
      fore(i,row,n-1)
        if(abs(a[i][col]) > abs(a[sel][col])) sel = i;
      if(abs(a[sel][col]) <= eps) continue;

      fore(i,col,m) swap (a[sel][i], a[row][i]);
      where[col] = row;

      forn(i,n){
        if (i != row) {
          int c = 1LL*a[i][col] * inv(a[row][col])%mod;
          for (int j=col; j<=m; ++j) a[i][j] = (mod + a[i][j] - (1LL*a[row][j] * c)%mod)%mod;
        }
      }
      ++row;
    }

    ans.assign(m, 0);
    forn(i,m){
      if(where[i] != -1) ans[i] = 1LL*a[where[i]][m] * inv(a[where[i]][i])%mod;
    }
    forn(i,n){
      ll sum = 0;
      forn(j,m) sum = (sum + 1LL*ans[j] * a[i][j])%mod;
      if(abs(sum - a[i][m]) > eps) return 0;
    }

    forn(i,m) if(where[i] == -1) return 1e9; /// infinitas soluciones
    return 1;
  }
```

6.21 Berlekamp Massey

```
// taken from https://codeforces.com/blog/entry/61306
struct ber_ma{
        vi BM(vi &x){
                vi ls,cur; int lf,ld;
                forn(i,sz(x)){
                        ll t=0;
                        forn(j,sz(cur)) t=(t+x[i-j-1]*(ll)cur[j])%mod;
                        if((t-x[i])%mod==0) continue;
                        if(!sz(cur)){
                                cur.resize(i+1);
                                lf=i; ld=(t-x[i])%mod;
                                continue;
                        }
                        ll k=-(x[i]-t)*inv(ld,mod);
                        vi c(i-lf-1); c.pb(k);
                        forn(j,sz(ls)) c.pb(-ls[j]*k%mod);
                        if(sz(c)<sz(cur)) c.resize(sz(cur));
                        forn(j,sz(cur)) c[j]=(c[j]+cur[j])%mod;
                        if(i-lf+sz(ls)>=sz(cur)) ls=cur,lf=i,ld=(t-x[i])%mod;
                        cur=c;
                }
                forn(i,sz(cur)) cur[i]=(cur[i]%mod+mod)%mod;
                return cur;
        }

        int m; //length of recurrence
        //a: first terms
        //h: relation
        vector<ll> a, h, t_, s, t;
        //calculate p*q mod f
        inline vector<ll> mull(vector<ll> p, vector<ll> q){
                forn(i,2*m) t_[i]=0;
                forn(i,m) if(p[i])
                        forn(j,m)
                                t_[i+j]=(t_[i+j]+p[i]*q[j])%mod;
                for(int i=2*m-1;i>=m;--i) if(t_[i])
                        forn(j,m)
                                t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%mod;
                forn(i,m) p[i]=t_[i];
                return p;
        }
        inline ll calc(ll k){
    if(k < sz(a)) return a[k];
                forn(i,m) s[i]=t[i]=0;
                s[0]=1;
                if(m!=1) t[1]=1;
                else t[0]=h[0];

                while(k){
                        if(k&1LL) s = mull(s,t);
                        t = mull(t,t); k/=2;
                }
                ll su=0;
                forn(i,m) su=(su+s[i]*a[i])%mod;
                return (su%mod+mod)%mod;
        }
        ber_ma(vi &x){
                vi v = BM(x); m=sz(v);
                h.resize(m), a.resize(m), s.resize(m);
                t.resize(m), t_.resize(2*m);
                forn(i,m) h[i]=v[i],a[i]=x[i];
        }
};
```

7 Dynamic Programming

7.1 Edit Distance

```
// O(m*n) donde cada uno es el tamano de cada string
int editDist(string &s1, string &s2){
  int m = sz(s1), n = sz(s2);
  int dp[m+1][n+1];
  forn(i,m+1)
    forn(j,n+1){
      if (i==0) dp[i][j] = j;
      else if (j==0) dp[i][j] = i;
      else if (s1[i-1] == s2[j-1]) dp[i][j] = dp[i-1][j
          -1];
      else dp[i][j] = 1 + min({dp[i][j-1],  // Insert
                              dp[i-1][j],  // Remove
                              dp[i-1][j-1]}); // Replace
    }
  return dp[m][n];
}
```

7.2 Longest common subsequence

```
const int nax = 1005;
int dp[nax][nax];
int lcs(const string &s, const string &t){
  int n = sz(s), m = sz(t);
  forn(j,m+1) dp[0][j] = 0;
  forn(i,n+1) dp[i][0] = 0;
  for1(i,n){
    for1(j,m){
      dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
      if (s[i-1] == t[j-1]){
        dp[i][j] = max(dp[i][j], dp[i-1][j-1] + 1);
      }
    }
  }
  return dp[n][m];
}
```

7.3 Longest increasing subsequence

```
// Complejidad n log n
int lis(const vi &a) {
  int n = a.size();
  vi d(n+1, inf);
  d[0] = -inf;

  for (int i = 0; i < n; i++) {
      int j = upper_bound(d.begin(), d.end(), a[i]) - d.
          begin();
      if (d[j-1] < a[i] && a[i] < d[j]) d[j] = a[i];
    }

    int ans = 0;
    for (int i = 0; i <= n; i++) {
      if (d[i] < inf) ans = i;
    }
    return ans;
}
```

7.4 Trick to merge intervals

```
// Option 1
for(int len= 0; len<n; ++len){
    for(int l= 0; l<n-len; ++l){
        int r= l+len;
        dp[l][r]= max(dp[l+1][r], dp[l][r-1]);
    }
}
// Option 2
for(int l= n-1; l>=0; --l){
    for(int r= l; r<n ; ++r){
        dp[l][r]= max(dp[l+1][r], dp[l][r-1]);
    }
}
```

7.5 Trick Sets DP

```
// Complexity O(N*2^N)
const int N;
int dp[1<<N][N+1];
int F[1<<N];
int A[1<<N];
// ith bit is ON S(mask, i) = S(mask, i-1)
// ith bit is OFF S(mask,i) = S(mask, i-1) + S(mask^(1<<i
    ), i-1)
//iterative version
forn(mask,(1<<N)){
        dp[mask][0] = A[mask];  //handle base case
            separately (leaf states)
        forn(i,N){
                if(mask & (1<<i))
                        dp[mask][i+1] = dp[mask][i] + dp[
                            mask^(1<<i)][i];
                else
                        dp[mask][i+1] = dp[mask][i];
        }
        F[mask] = dp[mask][N];
}
//memory optimized, super easy to code.
```

```
forn(i,(1<<N)) F[i] = A[i];
forn(i,N)
  forn(mask,(1<<N)){
    if(mask & (1<<i)) F[mask] += F[mask^(1<<i)];
  }
```

7.6 Divide and Conquer

```
const ll inf = 1e18;
const int nax = 1e3+20, kax = 20;

ll C[nax][nax], dp[kax][nax];
int n;

void compute(int k, int l, int r, int optl, int optr){
  if(l>r) return ;
  int mid= (l+r)/2, opt;
  pll best= {inf,-1};
  for(int i= max(mid,optl); i<= optr ; ++i ){
    best = min(best, {dp[k-1][i+1] + C[mid][i] ,i}  );
  }
  tie(dp[k][mid], opt) = best;
  compute(k,l, mid-1, optl, opt);
  compute(k,mid+1, r, opt, optr);
}

inside main(){
  fore(k,1,K) // definir el caso base k = 0.
    compute(k,0,n-1,0,n-1);
}
```

7.7 Knuth's Optimization

```
const int nax = 1e3+20;
const ll inf = LONG_LONG_MAX;
ll dp[nax][nax];
int k[nax][nax];
int C[nax][nax];  // puede depender de k

int main(){
  for(int len=2; len<n; ++len){
    for(int l=0; l< n-len; ++l){
      int r= l+len;
      ll &ans= dp[l][r];
      if(len== 2){
        k[l][r]= l+1;
        ans= C[l][r];
        continue;
      }
      ans= inf;
      for(int i= k[l][r-1]; i<= k[l+1][r]; ++i ){
        if(ans> dp[l][i]+ dp[i][r]){
          ans= dp[l][i] + dp[i][r];
          k[l][r]= i;
        }
      }
      ans+= C[l][r];
    }
  }
  cout<< dp[0][n-1]<<el;
}
```

7.8 Convex Hull Trick

```
struct line {
  ll m, b;
  ll eval(ll x) { return m * x + b; }
  ld inter(line &l) { return (ld) (b - l.b) / (l.m - m);
    }
};

struct cht {
  vector<line> lines;
  vector<ld> inter;
  int n;
  inline bool ok(line &a, line &b, line &c) {
    return a.inter(c) > a.inter(b);
  }
  void add(line &l) { /// m1 < m2 < m3 ...
    n = sz(lines);
    if(n && lines.back().m == l.m && lines.back().b >= l.
       b) return;
    if(n == 1 && lines.back().m == l.m && lines.back().b
       < l.b) lines.pop_back(), n--;
    while(n >= 2 && !ok(lines[n-2], lines[n-1], l)) {
      --n;
      lines.pop_back(); inter.pop_back();
    }
    lines.pb(l); n++;
    if(n >= 2) inter.pb(lines[n-2].inter(lines[n-1]));
  }
  ll get_max(ld x) {
    if(sz(lines) == 0) return LLONG_MIN;
    if(sz(lines) == 1) return lines[0].eval(x);
    int pos = lower_bound(all(inter), x) - inter.begin();
    return lines[pos].eval(x);
  }
};
```

7.9 CH Trick Dynamic

```
typedef ll T;
const T is_query=-(1LL<<62);
struct line {
        T m,b;
        mutable multiset<line>::iterator it,end;
```

```
        const line *succ(multiset<line>::iterator it)
            const {
                return (++it == end ? nullptr : &*it);
        }
        bool operator < (const line &rhs) const {
                if(rhs.b!=is_query) return m < rhs.m;
                const line *s = succ(it);
                if(!s) return 0;
                return b-s->b < (ld)(s->m-m)*rhs.m;
        }
};
struct CHT : public multiset<line> {
        bool bad(iterator y) {
                iterator z = next(y);
                if(y==begin()) {
                        if(z==end())return false;
                        return y->m == z->m && y->b <= z
                            ->b;
                }
                iterator x = prev(y);
                if(z == end()) return y->m==x->m&&y->b==x
                    ->b;
                return (ld)(x->b-y->b)*(z->m-y->m)>=(ld)(
                    y->b-z->b)*(y->m-x->m);
        }
        iterator next(iterator y){return ++y;}
        iterator prev(iterator y){return --y;}
        void add(T m, T b) {
                iterator y = insert((line){m,b});
                y->it = y; y->end = end();
                if(bad(y)){ erase(y); return; }
                while(next(y)!=end() && bad(next(y)))
                    erase(next(y));
                while(y!=begin() && bad(prev(y)))erase(
                    prev(y));
        }
        T eval(T x) { /// max
                line l = *lower_bound((line){x,is_query})
                    ;
                return l.m*x+l.b;
        }
};
```

8 Geometry

8.1 Point

```
struct pt {  // for 3D add z coordinate
        ld x,y;
        pt(ld x, ld y):x(x),y(y){}
        pt(){}
        ld norm2(){return *this**this;}
        ld norm(){return sqrt(norm2());}
        bool operator==(pt p){return abs(x-p.x)<=eps&&abs
            (y-p.y)<=eps;}
        bool operator!=(pt p){ return !operator==(p);}
        bool operator<(pt p)const{ // for convex hull/set
            /map
                return x<p.x-eps||(abs(x-p.x)<=eps&&y<p.y
                    -eps);}
        pt operator+(pt p){return pt(x+p.x,y+p.y);}
        pt operator-(pt p){return pt(x-p.x,y-p.y);}
        pt operator*(ld t){return pt(x*t,y*t);}
        pt operator/(ld t){return pt(x/t,y/t);}
        ///DOT
        ld operator*(pt p){return x*p.x+y*p.y;}
//      pt operator%(pt p){ // 3D
//              return pt(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y
    *p.x);}
        ld angle(pt p){ ///[0, pi]
                ld co = *this*p/(norm()*p.norm());
                return acos(max(-1.0L, min(1.0L, co)));
  }
        pt unit(){return *this/norm();}
        ld operator%(pt p){return x*p.y-y*p.x;}
        /// 2D from now on
        bool left(pt p, pt q){ // is it to the left of
            directed line pq?
                return (q-p)%(*this-p)>eps;
  }
        int left_int(pt p, pt q){ // is it to the left of
            directed line pq?
                ld cro = (q-p)%(*this-p);
                if(cro < eps)
      return -1;
                else
      return (abs(cro) <= eps ? 0 : 1);
  }
        pt rot(pt r){return pt(*this%r,*this*r);}
        pt rot(ld a){return rot(pt(sin(a),cos(a)));}
        pt rotp(ld a, pt p){
    pt aux = (*this - p).rot(a);
    return aux + p;
  }
};

pt ccw90(1,0), cw90(-1,0);

int sgn(ld x){
  if(x<0)
    return -1;
  else if(abs(x) <= eps)
    return 0;
  else
    return 1;
}

//pt zero = pt(0,0,0); //for 3D

ld orient(pt a, pt b, pt c){  ///C: >0 left, ==0 on AB,
```

```
    <0 right
  return (b-a)%(c-a);
}

ld small_angle(pt p, pt q){   ///[0, pi] ([0, 180])
  return acos(max(-1.0L, min((p*q)/(p.norm()*q.norm()), 1.0L)));
}

ld dir_ang_CW(pt a, pt b, pt c){ ///Vertex = B,  from BA -> BC (CW)
  if(orient(a,b,c) >= 0){
    return small_angle(a-b, c-b);
  } else{
    return 2*pi - small_angle(a-b, c-b);
  }
}

ld dir_ang_CCW(pt a, pt b, pt c){ ///Vertex = B,  from BA -> BC (CCW)
  if(orient(a,b,c) <= 0){
    return small_angle(a-b, c-b);
  } else{
    return 2*pi - small_angle(a-b, c-b);
  }
}

bool in_angle_CW(pt a, pt b, pt c, pt p){    ///is P inside ang(ABC) CW
  return dir_ang_CW(a, b, p) <= dir_ang_CW(a, b, c);
}

bool in_angle_CCW(pt a, pt b, pt c, pt p){    ///is P inside ang(ABC) CW
  return dir_ang_CCW(a, b, p) <= dir_ang_CCW(a, b, c);
}

///3D - ORIENT
ld orient(pt p, pt q, pt r, pt s){  return (q-p)%(r-p)|(s-p);}
bool coplanar(pt p, pt q, pt r, pt s){
  return abs(orient(p, q, r, s)) < eps;
}
bool skew(pt p, pt q, pt r, pt s){        ///skew := neither intersecting/parallel
  return abs(orient(p, q, r, s)) > eps;   ///lines: PQ, RS
}
ld orient_norm(pt p, pt q, pt r, pt n){   ///n := normal to a given plane PI
  return (q-p)%(r-p)|n;/// equivalent to 2D cross on PI (of ortogonal proj)
}
```

8.2 Line

```
int sgn2(ld x){return x<0?-1:1;}
struct ln {
        pt p, pq; //POINT + DIRECTION
        ln(pt p, pt q) : p(p), pq(q-p){}
        ln(ld a, ld b, ld c) : p(b == 0 ? pt(-c/a, 0) : pt(0, -c/b)), pq(pt(b, -a)){} ///ax + by + c = 0
        ln(){}
        bool has(pt r){return dist(r)<=eps;} ///check if point belongs
        bool seghas(pt r){return has(r)&&(r-p)*(r-(p+pq))<=eps;} ///check if point belongs to segment PQ
//      bool operator /(ln l){return (pq.unit()^l.pq.unit()).norm()<=eps;} // 3D
        bool operator/(ln l){return abs(pq.unit()%l.pq.unit())<=eps;} /// PARALLEL CHECK
        bool operator==(ln l){return *this/l&&has(l.p);}
        pt operator^(ln l){ /// intersection ln-ln
                if(*this/l) return pt(inf,inf);
                pt r=l.p+l.pq*((p-l.p)%pq/(l.pq%pq));
//              if(!has(r)){return pt(NAN,NAN,NAN);} // check only for 3D
                return r;
        }
        ld angle(ln l){return pq.angle(l.pq);}   ///angle bet. 2 lines
        int side(pt r){return has(r)?0:sgn2(pq%(r-p));} /// 1=L, 0= on, -1=R
        pt proj(pt r){return p+pq*((r-p)*pq/pq.norm2());}
        pt refl(pt r){return proj(r)*2-r;}
        ld dist(pt r){return (r-proj(r)).norm();}
        ld dist2(pt r){return (r - proj(r)).norm2();}
//      ls dist(ln l){ // only 3D
//              if(*this/l)return dist(l.p);
//              return abs((l.p-p)*(pq^l.pq))/(pq^l.pq).norm();
//      }
        ln rot(auto a){return ln(p,p+pq.rot(a));} /// 2D respecto a P
  ln perp_at(pt r){return ln(r, r+pq.rot(ccw90));}
  bool cmp_proj(pt r, pt s){return pq*r<pq*s;}
  int cmp_int(pt r, pt s){
    if(pq*r < pq*s)
      return -1;
    else
      return (pq*r == pq*s ? 0 : 1);
  }
  ln trans(pt d){ return ln(p + d, pq);}    ///d = dir. vec
};

ln bisec(ln l, ln m){ /// angle bisector
        pt p=l^m;
        return ln(p, p+l.pq.unit()+m.pq.unit());
}
```

```
ln bisec(pt p, pt q){ /// segment bisector (2D) (
    mediatriz)
        return ln((p+q)*.5,p).rot(ccw90);
}

///Segments
bool in_disk(pt a, pt b, pt p){return (a-p)*(b-p) <= 0;}
bool on_seg(pt a, pt b, pt p){ return orient(a,b,p) == 0
    && in_disk(a,b,p);}
bool proper_inter(pt a, pt b, pt c, pt d, pt &out){
  ld oa = orient(c,d,a),
     ob = orient(c,d,b),
     oc = orient(a,b,c),
     od = orient(a,b,d);
  if(oa*ob<0 && oc*od<0){
    out = (a*ob - b*oa) / (ob-oa);
    return true;
  }
  return false;
}
struct cmpX {
  bool operator()(pt a, pt b){
    return make_pair(a.x, a.y) < make_pair(b.x, b.y);
  }
};

set<pt,cmpX> seg_inter(pt a, pt b, pt c, pt d){ ///AB, CD
  pt out;
  if(proper_inter(a,b,c,d,out)) return {out};
  set<pt,cmpX> s;
  if(on_seg(c,d,a)) s.insert(a);
  if(on_seg(c,d,b)) s.insert(b);
  if(on_seg(a,b,c)) s.insert(c);
  if(on_seg(a,b,d)) s.insert(d);
  return s;
}

ld seg_pt(pt a, pt b, pt p){  ///DISTANTCE FROM P -> SEG.
    AB
  if(a!=b){
    ln l(a,b);
    /// a <= proj(p) && proj(p) <= b
    if(l.cmp_proj(a,p) && l.cmp_proj(p,b)) return l.dist(
        p);
  }
  return min((p-a).norm(), (p-b).norm());
}

ld seg_seg(pt a, pt b, pt c, pt d){  ///DISTANCE FROM SEG
    . AB -> SEG. CD
  pt aux;
  if(proper_inter(a,b,c,d,aux)) return 0;
  return min({seg_pt(a,b,c),seg_pt(a,b,d),seg_pt(c,d,a),
      seg_pt(c,d,b)});
}

//ld dist(ln l1, ln3 l2){      ///for 3D
//  p3 n = l1.d%l2.d;
//  if(n == zero){  ///parallel
//    return l1.dist(l2.o);
//  }
//  return abs((l2.o - l1.o)|n)/abs(n);
//}

///AGREGAR COMO ENCONTRAR LOS PUNTOS
```

8.3 Convex Hull

```
// CCW order
// Includes collinear points (change sign of EPS in left
    to exclude)
vector<pt> chull(vector<pt> p){
        if(sz(p)<3)return p;
        vector<pt> r;
        sort(all(p)); // first x, then y
        forn(i,sz(p)){ // lower hull
                while(sz(r) >=2 && r.back().left(r[sz(r)
                    -2], p[i]))
    r.pop_back();
                r.pb(p[i]);
        }
        r.pop_back();
        int k = sz(r);
        for(int i = sz(p)-1 ; i>=0 ; --i){ // upper hull
                while(sz(r) >= k+2 && r.back().left(r[sz(
                    r)-2], p[i]))
    r.pop_back();
                r.pb(p[i]);
        }
        r.pop_back();
        return r;
}
```

8.4 Polygon

```
int sgn(double x){return x<-eps?-1:x>eps;}
struct pol {
        int n; vector<pt> p;
        pol(){}
        pol(vector<pt> _p){p=_p;n = sz(p);}

        double area(){
                double r=0.;
                forn(i,n) r += p[i]%p[(i+1)%n];
                return abs(r)/2; // negative if CW,
                    positive if CCW
        }

        bool isConvex() {
    bool pos=false, neg=false;
```

```
  for (int i=0; i<n; i++) {
    int o = orient(p[i], p[(i+1)%n], p[(i+2)%n]);
    if (o > 0) pos = true;
    if (o < 0) neg = true;
  }
  return !(pos && neg);
}

    pt centroid(){ // (barycenter)
            pt r(0,0); double t=0;        ///REVISAR
            forn(i,n){
                    r = r+(p[i]+p[(i+1)%n])*(p[i]%p[(
                       i+1)%n]);
                    t += p[i]%p[(i+1)%n];
            }
            return r/t/3;
    }

    bool has(pt q){ /// O(n)
            forn(i,n)
   if(ln(p[i],p[(i+1)%n]).seghas(q))
     return true; //lies on a segment
            int cnt=0;
            forn(i,n){
                    int j = (i+1)%n;
                    int k = sgn((q-p[j])%(p[i]-p[j]))
                       ;
                    int u = sgn(p[i].y - q.y),v=sgn(p
                       [j].y - q.y);
                    if(k>0 && u<0 && v>=0)cnt++;
                    if(k<0 && v<0 && u>=0)cnt--;
            }
            return cnt!=0;
    }

    void remove_col(){      ///for haslog
  vector<pt> s;
  forn(i, n){
    line l(p[(i-1+n)%n], p[(i+1)%n]);
    if (!l.seghas(p[i])) s.pb(p[i]);
  }
  p.swap(s);
    }

    void normalize(){ /// (call before haslog, remove
         collinear first)
            remove_col();
            if(p[2].left(p[0],p[1])) reverse(all(p));
            int pi = min_element(all(p)) - p.begin();
            vector<pt> s(n);
            forn(i,n) s[i] = p[(pi+i)%n];
            p.swap(s);
    }

    bool haslog(pt q){ /// O(log(n)) only CONVEX.
        Call normalize first
            if(q.left(p[0],p[1]) || q.left(p.back(),p
                    [0])) return false;
            int a=1, b=sz(p)-1;    // returns true if
                 point on boundary
            while(b-a > 1){          // (change sign
                of EPS in left
                    int c = (a+b)/2;       //  to
                        return false in such case)
                    if(!q.left(p[0],p[c])) a=c;
                    else b=c;
            }
            return !q.left(p[a],p[a+1]);
    }

    pt farthest(pt v){ /// O(log(n)) only CONVEX
            if(n < 10){
                    int k=0;
                    for1(i,n-1) if(v*(p[i]-p[k]) >
                        eps) k=i;
                    return p[k];
            }
            if(n == sz(p)) p.pb(p[0]);
            pt a = p[1]-p[0];
            int s=0, e=n, ua=v*a>eps;
            if(!ua && v*(p[n-1]-p[0]) <= eps) return
                p[0];
            while(1){
                    int m = (s+e)/2; pt c=p[m+1]-p[m
                        ];
                    int uc = v*c> eps;
                    if(!uc && v*(p[m-1]-p[m]) <= eps)
                         return p[m];
                    if(ua && (!uc||v*(p[s]-p[m]) >
                        eps))e=m;
                    else if(ua || uc || v*(p[s]-p[m])
                         >= -eps) s=m, a=c, ua=uc;
                    else e=m;
                    assert(e>s+1);
            }
    }

    pol cut(ln l){   // cut CONVEX polygon by line l
            vector<pt> q;  // returns part at left of
                 l.pq
            forn(i,n){
                    int d0 = sgn(l.pq%(p[i]-l.p)), d1
                         = sgn(l.pq%(p[(i+1)%n]-l.p));
                    if(d0 >= 0) q.pb(p[i]);
                    ln m(p[i], p[(i+1)%n]);
                    if(d0*d1<0 && !(l/m)) q.pb(l^m);
            }
            return pol(q);
    }
    ld intercircle(circle c){ /// area of
        intersection with circle
            ld r = 0.;
```

```
            forn(i,n){
                int j = (i+1)%n; ld w = c.intertriangle(p[i], p[j]);
                if((p[j]-c.o)%(p[i]-c.o) > 0) r+=w;
                else r-=w;
            }
            return abs(r);
        }

        ld callipers(){ // square distance: pair of most distant points
            ld r=0;     // prereq: convex, ccw, NO COLLINEAR POINTS
            for(int i=0,j=n<2?0:1; i<j; ++i){
                for(;;j=(j+1)%n){
                    r = max(r,(p[i]-p[j]).norm2());
                    if((p[(i+1)%n]-p[i])%(p[(j+1)%n]-p[j]) <= eps)
                        break;
                }
            }
            return r;
        }
};
// Dynamic convex hull trick
vector<pol> w;
void add(pt q){ // add(q), O(log^2(n))
        vector<pt> p = {q};
        while(sz(w) && sz(w.back().p) < 2*sz(p)){
                for(pt v: w.back().p) p.pb(v);
                w.pop_back();
        }
        w.pb(pol(chull(p)));
}
ll query(pt v){ // max(q*v:q in w), O(log^2(n))
        ll r = -inf;
        for(auto& p : w) r = max(r, p.farthest(v)*v);
        return r;
}

/// max_dist between 2 points (pa, pb) of 2 conv. polygons (a,b)
ll rot_cal(vector<pt>& a, vector<pt>& b){
  pair<ll, int> start = {-1, -1};
  if(sz(a) == 1) swap(a, b);

  forn(i, sz(a)){
    start = max(start, {(b[0] - a[i]).norm2(), i});
  }

  if(sz(b) == 1) return start.fi;

  ll r = 0;
  for(int i = 0, j = start.se; i<sz(b); ++i){
    for(;;j=(j+1)%sz(a)){
        r = max(r,(b[i]-a[j]).norm2());
        if((b[(i+1)%sz(b)]-b[i])%(a[(j+1)%sz(a)]-a[j]) <= eps) break;
    }
  }
  return r;
}
```

8.5 Circle

```
struct circle {
        pt o; ld r;
        circle(pt o, ld r):o(o),r(r){}
        circle(pt x, pt y, pt z){o=bisec(x,y)^bisec(x,z);r=(o-x).norm();}///circumcircle
        bool has(pt p){return (o-p).norm()<=r+eps;}
        vector<pt> operator^(circle c){ // ccw
                vector<pt> s;
                ld d = (o - c.o).norm();
                if(d > r+c.r+eps || d+min(r,c.r)+eps < max(r,c.r)) return s;
                ld x = (d*d - c.r*c.r + r*r)/(2*d);
                ld y = sqrt(r*r - x*x);
                pt v = (c.o - o)/d;
                s.pb(o + v*x - v.rot(ccw90)*y);
                if(y>eps) s.pb(o + v*x + v.rot(ccw90)*y);
                return s;
        }
        vector<pt> operator^(ln l){
                vector<pt> s;
                pt p=l.proj(o);
                ld d=(p-o).norm();
                if(d-eps>r)return s;
                if(abs(d-r)<=eps){s.pb(p);return s;}
                d=sqrt(r*r-d*d);
                s.pb(p+l.pq.unit()*d);
                s.pb(p-l.pq.unit()*d);
                return s;
        }
        vector<pt> tang(pt p){
                ld d = sqrt((p-o).norm2()-r*r);
                return *this^circle(p,d);
        }
        bool in(circle c){ // non strict
                ld d = (o-c.o).norm();
                return d+r <= c.r+eps;
        }
        ld intertriangle(pt a, pt b){ // area of intersection with oab
                if(abs((o-a)%(o-b))<=eps)return 0.;
                vector<pt> q = {a}, w = *this^ln(a,b);
                if(sz(w)==2) for(auto p:w) if((a-p)*(b-p)<-eps) q.pb(p);
                q.pb(b);
```

```
            if(sz(q)==4 && (q[0]-q[1])*(q[2]-q[1])>
                eps) swap(q[1],q[2]);
            ld s=0;
            forn(i, sz(q)-1){
                if(!has(q[i]) || !has(q[i+1])) s
                    +=r*r*(q[i]-o).angle(q[i+1]-o)
                    /2;
                else s+=abs((q[i]-o)%(q[i+1]-o)
                    /2);
            }
            return s;
        }
};

vector<ld> intercircles(vector<circle> c){
        vector<ld> r(sz(c)+1); // r[k]: area covered by
            at least k circles
        forn(i,sz(c)){          // O(n^2 log n) (high
            constant)
            int k=1; Cmp s(c[i].o);
            vector<pair<pt,int> > p =
            {{c[i].o+pt(1,0)*c[i].r,0},{c[i].o-pt
                (1,0)*c[i].r,0}};
            forn(j, sz(c))if(j!=i){
                bool b0 = c[i].in(c[j]), b1=c[j].
                    in(c[i]);
                if(b0 && (!b1||i<j))k++;
                else if(!b0 && !b1){
                    auto v = c[i]^c[j];
                    if(sz(v)==2){
                        p.pb({v[0],1});
                        p.pb({v[1],-1});
                        if(s(v[1],v[0]))k
                            ++;
                    }
                }
            }
            sort(p.begin(),p.end(),
                [&](pair<pt,int> a, pair<pt,int>
                    b){return s(a.fi,b.fi);});
            forn(j,sz(p)){
                pt p0 = p[j?j-1:sz(p)-1].fi, p1 =
                     p[j].fi;
                ld a=(p0-c[i].o).angle(p1-c[i].o)
                    ;
                r[k] += (p0.x-p1.x)*(p0.y+p1.y)
                    /2+c[i].r*c[i].r*(a-sin(a))/2;
                k+=p[j].se;
            }
        }
        return r;
}
```

8.6 Radial Order

```
struct Cmp { // IMPORTANT: add const in pt operator -
        pt r;
        Cmp(pt r):r(r){}
        int cuad(const pt &a)const {
                if(a.x>0 && a.y>=0) return 0;
                if(a.x<=0 && a.y>0) return 1;
                if(a.x<0 && a.y<=0) return 2;
                if(a.x>=0 && a.y<0) return 3;
                assert(a.x==0&&a.y==0);
                return -1;
        }
        bool cmp(const pt& p1, const pt& p2)const {
                int c1=cuad(p1), c2=cuad(p2);
                if(c1==c2) return p1.y*p2.x < p1.x*p2.y;
                return c1<c2;
        }
        bool operator()(const pt& p1, const pt& p2)const
            {
                return cmp(p1-r,p2-r);
        }
};
```

8.7 Coords

```
struct coords{
  p3 o, dx, dy, dz;
  coords(){}
  coords(p3 p, p3 q, p3 r){         ///Creates a R3 space
      with X-Y plane at PQR
    o = p;
    dx = unit(q-p);
    dz = unit(dx%(r-p));
    dy = dz%dx;
  }

  pt pos2(p3 p){ return pt((p-o)*dx, (p-o)*dy);}
                ///2D Coordinates of the orthogonal proj
  p3 pos3(p3 p){ return p3((p-o)*dx, (p-o)*dy, (p-o)*dz)
      ;}  ///3D Coordinates in new R3-space
};
```

8.8 Plane

```
struct plane {
        pt a, n; // n: normal unit vector
        plane(pt a, pt b, pt c): a(a),n(((b-a)^(c-a)).
            unit()){}
        plane(){}
        bool has(pt p){return abs((p-a)*n) <= eps;}
                    ///DONE
        double angle(plane w){return acos(n*w.n);}
        double dist(pt p){return abs((p-a)*n);}
                        ///DONE
```

```
        bool inter(ln l, pt& r){   //LINE-PLANE --> POINT
                double x = n*(l.p+l.pq-a), y = n*(l.p-a);
                if(abs(x-y) <= eps) return false;
                r = (l.p*x-(l.p+l.pq)*y)/(x-y);
                return true;
        }
        bool inter(plane w, ln& r){//PLANE-PLANE --> LINE
                pt nn = n^w.n;pt v = n^nn;double d = w.n*
                   v;
                if(abs(d) <= eps) return false;
                pt p = a+v*(w.n*(w.a-a)/d);
                r = ln(p,p+nn);
                return true;
        }
        pt proj(pt p){inter(ln(p,p+n),p); return p;}
                        ///DONE
};
```

8.9 Halfplane

```
// polygon intersecting left side of hps
struct hp: public ln{
        ld angle;
        hp(){}
        hp(pt a, pt b){p=a; pq=b-a; angle=atan2(pq.y,pq.x
           );}
        bool operator<(hp b)const{return angle<b.angle;}
        bool out(pt q){return pq%(q-p)<-eps;}
};

vector<pt> intersect(vector<hp> b){
        vector<pt>bx = {{inf,inf},{-inf,inf},{-inf,-inf
           },{inf,-inf}};
        forn(i,4) b.pb(hp(bx[i],bx[(i+1)%4]));
        sort(all(b));
        int n=sz(b), q=1, h=0;
        vector<hp> c(sz(b)+10);
        forn(i,n){
                while(q<h&&b[i].out(c[h]^c[h-1])) h--;
                while(q<h&&b[i].out(c[q]^c[q+1])) q++;
                c[++h]=b[i];
                if(q<h&&abs(c[h].pq%c[h-1].pq)<eps){
                        if(c[h].pq*c[h-1].pq<=0) return
                           {};
                        h--;
                        if(b[i].out(c[h].p)) c[h]=b[i];
                }
        }
        while(q<h-1&&c[q].out(c[h]^c[h-1]))h--;
        while(q<h-1&&c[h].out(c[q]^c[q+1]))q++;
        if(h-q<=1)return {};
        c[h+1]=c[q];
        vector<pt> s;
        fore(i,q,h) s.pb(c[i]^c[i+1]);
                return s;
}
```

8.10 Sphere

```
p3 sph_deg(ld r, ld lat, ld lon){
  lat *= pi/180, lon *= pi/180;
  return p3(r*cos(lat)*cos(lon), r*cos(lat)*sin(lon), r*
     sin(lat));
}
p3 sph(ld r, ld lat, ld lon){
  return p3(r*cos(lat)*cos(lon), r*cos(lat)*sin(lon), r*
     sin(lat));
}

int sph_ln_inter(p3 o, ld r, ln3 l, pair<p3,p3> &out){
  ld h2 = r*r - l.sq_dist(o);
  if(h2 < 0) return 0;  ///no intersection
  p3 p = l.proj(o);
  p3 h = l.d*sqrt(h2)/abs(l.d);   ///media cuerda
  out = {p-h, p+h};
  return 1 + (h2>0);
}

ld sph_dist(p3 o, ld r, p3 a, p3 b){    ///if A,B not in
   sphere -> takes radial projections (from O)
  return r * angle(a-o, b-o);
}

bool valid_seg(p3 a, p3 b){        ///Accepts A==B
  return a%b != zero || (a*b) > 0; ///Denies opposite to
     each other (seg not well defined)
}

bool proper_inter(p3 a, p3 b, p3 c, p3 d, p3 &out){
  p3 ab = a%b, cd = c%d;
  int oa = sgn(cd*a),
      ob = sgn(cd*b),
      oc = sgn(ab*c),
      od = sgn(ab*d);
  out = ab%cd * od;
  return (oa != ob && oc != od && oa != oc);
}

bool on_seg(p3 a, p3 b, p3 p){    ///segment = [A,B]
  p3 n = a%b;
  if(n == zero){
    return a%p == zero && (a*p) > 0;
  }
  return (n*p) == 0 && (n*(a%p)) >= 0 && (n*(b%p)) <= 0;
}

struct dir_set : vector<p3> {
  using vector::vector; ///import constructors
  void insert(p3 p){
```

```
    for(p3 q : *this)
      if(p%q == zero) return;
    pb(p);
  }
};

dir_set seg_inter(p3 a, p3 b, p3 c, p3 d){
  assert(valid_seg(a, b) && valid_seg(c, d));
  p3 out;
  if(proper_inter(a, b, c, d, out)) return {out};
  dir_set s;
  if(on_seg(c, d, a)) s.insert(a);
  if(on_seg(c, d, b)) s.insert(b);
  if(on_seg(a, b, c)) s.insert(c);
  if(on_seg(a, b, d)) s.insert(d);
  return s;
}

ld angle_sph(p3 a, p3 b, p3 c){
  return angle(a%b, a%c);
}

ld oriented_angle_sph(p3 a, p3 b, p3 c){
  if((a%b*c) >= 0)
    return angle_sph(a, b, c);
  else
    return 2*pi - angle_sph(a, b, c);
}

ld area_sph(ld r, vector<p3> &p){       ///for solid
   angle -> r = 1
  int n = sz(p);
  ld sum = -(n-2)*pi;
  forn(i,n)
    sum += oriented_angle_sph(p[(i+1)%n], p[(i+2)%n], p[i
      ]);
  return r*r*sum;
}

int winding_number(vector<vector<p3>> &faces){
  ld sum = 0;
  for(vector<p3> f: faces)
    sum += remainder(area_sph(1, f), 4*pi);
  return round(sum / (4*pi));
}
```

8.11 Polih

```
p3 vec_area2(vector<p3> &p){  ///normal vector to the
   surface
  p3 a = zero;
  forn(i, n)
    a = a + p[i]%p[(i+1)%sz(p)];
  return a;
}
ld area(vector<p3> &p){
  return abs(vec_area2(p))/2.0;
}

struct edge{
  int v;
  bool same;    ///:= is common edge in same order?
};

void reorient(vector<vector<p3>> &faces){ ///given a
   series of faces, make orientations (normal vec's)
   consistent
  int n = sz(faces);
  ///Find common edges + create adjacency graph
  vector<vector<edge>> g(n);
  map<pair<p3,p3>, int> es;
  forn(u, n){
    int m = sz(faces[u]);
    forn(i, m){
      p3 a = faces[u][i], b = faces[u][(i+1)%m];
      ///let's look at edge [a, b];
      if(es.count({a,b})){           ///seen in same order
        int v = es[{a,b}];
        g[u].pb({v, true});
        g[v].pb({u, true});
      } else if(es.count({b,a})){  ///seen in diff order
        int v = es[{b,a}];
        g[u].pb({v, false});
        g[v].pb({u, false});
      } else{                        ///not seen yet
        es[{a,b}] = u;
      }
    }
  }
  ///bfs to find which faces should be flipped
  vector<bool> seen(n, false), flip(n);
  flip[0] = false;
  queue<int> q;  q.push(0);
  while(sz(q)){
    int u = q.front();  q.pop();
    for(edge e: g[u]){
      if(seen[e.v]) continue;
      seen[e.v] = true;
      flip[e.v] = flip[u]^e.same; ///If the edge was in
         same order
      q.push(e.v);                ///one of the two
         should be flipped
    }
  }
  ///perform the flips
  forn(i, n){
    if(flip[i])
      reverse(all(faces[u]));
  }
}

ld volume(vector<vector<p3>> &faces){
```

```
  ld vol = 0;
  for(vector<p3> f: faces){
    vol += (vec_area2(f)*f[0]);
  }
  return abs(vol)/6.0;        ///could be <0 if normals
     point to inside
}
```

8.12 Line3

```
struct ln3{   ///Remove "3" if not 2D needed
  p3 d, o;    ///direction, origin
  ln3(){}
  ln3(p3 p, p3 q) : d(q-p), o(p){}    ///given 2 points
  ln3(plane p1, plane p2){            ///given 2 planes
    d = p1.n%p2.n;
    o = (p2.n%p1.d - p1.n%p2.d)%d/sq(d);
  }
  ld sq_dist(p3 p){ return sq(d%(p-o)/sq(d));}
  ld dist(p3 p){ return sqrt(sq_dist(p));}
  bool cmp_proj(p3 p1, p3 p2){ return (d|p1) < (d|p2);}
  int cmp(p3 p, p3 q){
    if((d|p) < (d|q)){
      return -1;
    } else if((d|p) == (d|q)){
      return 0;
    } else{
      return 1;
    }
  }
  p3 proj(p3 p){ return o + d*(d|(p-o))/sq(d);}
  p3 refl(p3 p){ return proj(p)*2 - p;}
  p3 inter(plane p){ return o - d*p.side(o)/(d|p.n);}
};
///LN3 - LN3
ld dist(ln3 l1, ln3 l2){
  p3 n = l1.d%l2.d;
  if(n == zero){  ///parallel
    return l1.dist(l2.o);
  }
  return abs((l2.o - l1.o)|n)/abs(n);
}

p3 closest(ln3 l1, ln3 l2){    ///closest point ON line
    l1 TO line l2
  p3 n2 = l2.d%(l1.d%l2.d);
  return l1.o + l1.d*((l2.o-l1.o)|n2)/(l1.d|n2));
}
ld angle (ln3 l1, ln3 l2){
  return small_angle(l1.d, l2.d);
}
bool parallel(ln3 l1, ln3 l2){
  return l1.d%l2.d == zero;
}
bool perp(ln3 l1, ln3 l2){
  return (l1.d|l2.d) == 0;
}
///PLANE - LN3
ld angle(plane p, ln3 l){
  return pi/2 - small_angle(p.n, l.d);
}
bool parallel(plane p, ln3 l){
  return (p.n|l.d) == 0;
}
bool perp(plane p, ln3 l){
  return p.n%l.d == zero;
}
ln3 perp_at(plane p, p3 o){
  return line(o, o+p.n);
}
plane perp_at(ln3 l, p3 o){
  return plane(l.d, o);
}
```

8.13 Point3

```
struct p3{
  ld x, y, z;
  p3() {}
  p3 (ld xx, ld yy, ld zz){x = xx, y = yy, z = zz;}
  ///scalar operators
  p3 operator*(ld f){ return p3(x*f, y*f, z*f);}
  p3 operator/(ld f){ return p3(x/f, y/f, z/f);}
  ///p3 operators
  p3 operator-(p3 p){ return p3(x-p.x, y-p.y, z-p.z);}
  p3 operator+(p3 p){ return p3(x+p.x, y+p.y, z+p.z);}
  p3 operator%(p3 p){ return p3(y*p.z - z*p.y, z*p.x - x*
      p.z, x*p.y - y*p.x);}    /// (|p||q|sin(ang))*
      normal
  ld operator|(p3 p){ return x*p.x + y*p.y + z*p.z;}
  ///Comparators
  bool operator==(p3 p){ return tie(x,y,z) == tie(p.x,p.y
      ,p.z);}
  bool operator!=(p3 p){ return !operator==(p);}
  bool operator<(p3 p){ return tie(x,y,z) < tie(p.x,p.y,p
      .z);}
};
p3 zero = p3(0,0,0);

template <typename T> int sgn(T x){
  return (T(0) < x) - (x < T(0));
}

///BASICS
ld sq(p3 p){  return p|p;}
```

```cpp
ld abs(p3 p){ return sqrt(sq(p));}
p3 unit(p3 p) { return p/abs(p);}

///ANGLES
ld angle(p3 p, p3 q){          ///[0, pi]
  ld co = (p|q)/abs(p)/abs(q);
  return acos(max(-1.0, min(1.0, co)));
}
ld small_angle(p3 p, p3 q){   ///[0, pi/2]
  return acos(min(abs(p|q)/abs(p)/abs(q), 1.0))
}

///3D - ORIENT
ld orient(p3 p, p3 q, p3 r, p3 s){  return (q-p)%(r-p)|(s-p);}
bool coplanar(p3 p, p3 q, p3 r, p3 s){
  return abs(orient(p, q, r, s)) < eps;
}
bool skew(p3 p, p3 q, p3 r, p3 s){         ///skew := neither intersecting/parallel
  return abs(orient(p, q, r, s)) > eps;    ///lines: PQ, RS
}
ld orient_norm(p3 p, p3 q, p3 r, p3 n){    ///n := normal to a given plane PI
  return (q-p)%(r-p)|n;/// equivalent to 2D cross on PI (of ortogonal proj)
}
```

8.14 Plane3

```cpp
struct plane{   ///ax + by + cz = d <=> n|(x, y, z) = d (all points with p|n = d)
  p3 n; ld d;   ///normal, offset
  plane() {}
  plane(p3 n, ld d) : n(n), d(d) {}
  plane(p3 n, p3 p) : n(n), d(n|p) {}                       ///given normal, point
  plane(p3 p, p3 q, p3 r) : plane((q-p)%(r-p), p) {}  ///given 3 points (uses previous line!!)
  ///point operators
  ld side(p3 p){ return (n|p) - d;}     ///>0 side pointed by n (above), ==0 on plane, <0 below
  ld dist(p3 p){ return abs(side(p))/abs(n);}
  bool has(pt p){return dist(p) <= eps;}
  p3 proj(p3 p){ return p - n*side(p)/sq(n);}
  p3 refl(p3 p){ return p - n*2*side(p)/sq(n);}
  ///translations
  plane translate(p3 t){ return plane(n, d+(n|t));}
  plane shift_up(ld dis) { return plane(n, d+dis*abs(n));}
};
```

```cpp
///PLANE - PLANE
ld angle(plane p1, plane p2){
  return small_angle(p1.n, p2.n);
}
bool parallel(plane p1, plane p2){
  return p1.n%p2.n == zero;
}
bool perp(plane p1, plane p2){
  return (p1.n|p2.n) == 0;
}

///PLANE - LN3
ld angle(plane p, ln3 l){
  return pi/2 - small_angle(p.n, l.d);
}
bool parallel(plane p, ln3 l){
  return (p.n|l.d) == 0;
}
bool perp(plane p, ln3 l){
  return p.n%l.d == zero;
}
ln3 perp_at(plane p, p3 o){
  return line(o, o+p.n);
}
plane perp_at(ln3 l, p3 o){
  return plane(l.d, o);
}
```

9 Miscellaneous

9.1 Counting Sort

```cpp
// it suppose that every element is non-negative
// in other case just translate to the right the elements
void counting_sort(vi &a){
  int n = sz(a);
  int maximo = *max_element(all(a));
  vector<int> cnt(maximo+1);
  forn(i,n) ++cnt[a[i]];
  for(int i = 0, j = 0; i <= maximo; ++i)
    while(cnt[i]--) a[j++] = i;
}
```

9.2 Expression Parsing

```cpp
bool delim(char c) {
  return c == ' ';
}
bool is_op(char c) {
  return c == '+' || c == '-' || c == '*' || c == '/';
}
```

```
bool is_unary(char c) {
  return c == '+' || c=='-';
}
int priority (char op) {
  if (op < 0) return 3; // unary operator
  if (op == '+' || op == '-') return 1;
  if (op == '*' || op == '/') return 2;
  return -1;
}
void process_op(stack<int>& st, char op) {
  if (op < 0) {
    int l = st.top(); st.pop();
    switch (-op) {
      case '+': st.push(l); break;
      case '-': st.push(-l); break;
    }
  } else {
    int r = st.top(); st.pop();
    int l = st.top(); st.pop();
    switch (op) {
      case '+': st.push(l + r); break;
      case '-': st.push(l - r); break;
      case '*': st.push(l * r); break;
      case '/': st.push(l / r); break;
    }
  }
}

int evaluate(string& s) {
  stack<int> st;
  stack<char> op;
  bool may_be_unary = true;
  forn(i,sz(s)) {
    if (delim(s[i]))
      continue;

    if (s[i] == '(') {
      op.push('(');
      may_be_unary = true;
    } else if (s[i] == ')') {
      while (op.top() != '(') {
        process_op(st, op.top());
        op.pop();
      }
      op.pop();
      may_be_unary = false;
    } else if (is_op(s[i])) {
      char cur_op = s[i];
      if (may_be_unary && is_unary(cur_op))
        cur_op = -cur_op;
      while (sz(op) && (
          (cur_op >= 0 && priority(op.top()) >= priority(
             cur_op)) ||
             (cur_op < 0 && priority(op.top()) >
                priority(cur_op))
             )) {
        process_op(st, op.top());
        op.pop();
      }
      op.push(cur_op);
      may_be_unary = true;
    } else {
      int number = 0;
      while (i < sz(s) && isalnum(s[i]))
        number = number * 10 + s[i++] - '0';
      --i;
      st.push(number);
      may_be_unary = false;
    }
  }

  while(sz(op)){
    process_op(st, op.top());
    op.pop();
  }
  return st.top();
}
```

9.3 Ternary Search

```
double ternary_search(double l, double r) {
  while (r - l > eps) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    double f1 = f(m1), f2 = f(m2);
    if (f1 < f2) l = m1;
    else r = m2;
  }
  return f(l);                    //return the maximum of
      f(x) in [l, r]
}
```

10 Theory

DP Optimization Theory

Name	Original Recurrence	Sufficient Condition	From	To
CH 1	$dp[i] = min_{j<i}\{dp[j] + b[j] * a[i]\}$	$b[j] \geq b[j+1]$ Optionally $a[i] \leq a[i+1]$	$O(n^2)$	$O(n)$
CH 2	$dp[i][j] = min_{k<j}\{dp[i-1][k] + b[k] * a[j]\}$	$b[k] \geq b[k+1]$ Optionally $a[j] \leq a[j+1]$	$O(kn^2)$	$O(kn)$
D&Q	$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j+1]$	$O(kn^2)$	$O(kn \log n)$
Knuth	$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$	$A[i, j-1] \leq A[i, j] \leq A[i+1, j]$	$O(n^3)$	$O(n^2)$

Notes:

- $A[i][j]$ - the smallest k that gives the optimal answer, for example in $dp[i][j] = dp[i-1][k] + C[k][j]$
- $C[i][j]$ - some given cost function
- We can generalize a bit in the following way $dp[i] = \min_{j<i}\{F[j] + b[j] * a[i]\}$, where $F[j]$ is computed from $dp[j]$ in constant time

Combinatorics

Sums

$\sum_{k=0}^{n} k = n(n+1)/2$

$\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$

$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$

$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$

$\sum_{k=0}^{n} k^4 = (6n^5 + 15n^4 + 10n^3 - n)/30$

$\sum_{k=0}^{n} k^5 = (2n^6 + 6n^5 + 5n^4 - n^2)/12$

$\sum_{k=0}^{n} x^k = (x^{n+1} - 1)/(x-1)$

$\sum_{k=0}^{n} kx^k = (x - (n+1)x^{n+1} + nx^{n+2})/(x-1)^2$

$1 + x + x^2 + \cdots = 1/(1-x)$

$\binom{n}{k} = \frac{n!}{(n-k)!k!}$

$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$

$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$

$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$

$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$

$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$

$12! \approx 2^{28.8}$

$20! \approx 2^{61.1}$

- Hockey-stick identity $\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$
- Number of ways to color n-objects with r-colors if all colors must be used at least once $\sum_{k=0}^{r} \binom{r}{k}(-1)^{r-k}k^n$ o $\sum_{k=0}^{r} \binom{r}{r-k}(-1)^k(r-k)^n$

Binomial coefficients

Number of ways to pick a multiset of size k from n elements: $\binom{n+k-1}{k}$

Number of n-tuples of non-negative integers with sum s: $\binom{s+n-1}{n-1}$, at most s: $\binom{s+n}{n}$

Number of n-tuples of positive integers with sum s: $\binom{s-1}{n-1}$

Number of lattice paths from $(0,0)$ to (a,b), restricted to east and north steps: $\binom{a+b}{a}$

Multinomial theorem. $(a_1 + \cdots + a_k)^n = \sum \binom{n}{n_1,\ldots,n_k} a_1^{n_1} \ldots a_k^{n_k}$, where $n_i \geq 0$ and $\sum n_i = n$.

$$\binom{n}{n_1,\ldots,n_k} = M(n_1,\ldots,n_k) = \frac{n!}{n_1!\ldots n_k!}$$

$$M(a,\ldots,b,c,\ldots) = M(a+\cdots+b,c,\ldots)M(a,\ldots,b)$$

Catalan numbers.

- $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$ con $n \geq 0$, $C_0 = 1$ y $C_{n+1} = \frac{2(2n+1)}{n+2}C_n$ $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$
- 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670
- C_n is the number of: properly nested sequences of n pairs of parentheses; rooted ordered binary trees with $n+1$ leaves; triangulations of a convex $(n+2)$-gon.

Derangements. Number of permutations of $n = 0, 1, 2, \ldots$ elements without fixed points is $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$ Recurrence: $D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$. Corollary: number of permutations with exactly k fixed points is $\binom{n}{k}D_{n-k}$.

Stirling numbers of 1^{st} kind. $s_{n,k}$ is $(-1)^{n-k}$ times the number of permutations of n elements with exactly k permutation cycles. $|s_{n,k}| = |s_{n-1,k-1}| + (n-1)|s_{n-1,k}|$. $\sum_{k=0}^{n} s_{n,k}\, x^k = x^{\underline{n}}$

Stirling numbers of 2^{nd} kind. $S_{n,k}$ is the number of ways to partition a set of n elements into exactly k non-empty subsets. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = 1$. $x^n = \sum_{k=0}^{n} S_{n,k}\, x^{\underline{k}}$

Bell numbers. B_n is the number of partitions of n elements. $B_0, \ldots = 1, 1, 2, 5, 15, 52, 203, \ldots$

$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k = \sum_{k=1}^{n} S_{n,k}$. Bell triangle: $B_r = a_{r,1} = a_{r-1,r-1}$, $a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

Bernoulli numbers. $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1}\sum_{k=0}^{n}\binom{n+1}{k}B_k m^{n+1-k}$.

$\sum_{j=0}^{m}\binom{m+1}{j}B_j = 0$. $B_0 = 1$, $B_1 = -\frac{1}{2}$. $B_n = 0$, for all odd $n \neq 1$.

Eulerian numbers. $E(n,k)$ is the number of permutations with exactly k descents $(i : \pi_i < \pi_{i+1})$ / ascents $(\pi_i > \pi_{i+1})$ / excedances $(\pi_i > i)$ / $k+1$ weak

excedances ($\pi_i \geq i$).
Formula: $E(n,k) = (k+1)E(n-1,k) + (n-k)E(n-1,k-1)$. $x^n = \sum_{k=0}^{n-1} E(n,k)\binom{x+k}{n}$.

Burnside's lemma. The number of orbits under group G's action on set X: $|X/G| = \frac{1}{|G|}\sum_{g \in G} |X_g|$, where $X_g = \{x \in X : g(x) = x\}$. ("Average number of fixed points.")
Let $w(x)$ be weight of x's orbit. Sum of all orbits' weights: $\sum_{o \in X/G} w(o) = \frac{1}{|G|}\sum_{g \in G}\sum_{x \in X_g} w(x)$.

Number Theory

Linear diophantine equation. $ax + by = c$. Let $d = \gcd(a,b)$. A solution exists iff $d|c$. If (x_0, y_0) is any solution, then all solutions are given by $(x,y) = (x_0 + \frac{b}{d}t, y_0 - \frac{a}{d}t)$, $t \in \mathbb{Z}$. To find some solution (x_0, y_0), use extended GCD to solve $ax_0 + by_0 = d = \gcd(a,b)$, and multiply its solutions by $\frac{c}{d}$.

Linear diophantine equation in n variables: $a_1x_1 + \cdots + a_nx_n = c$ has solutions iff $\gcd(a_1, \ldots, a_n)|c$. To find some solution, let $b = \gcd(a_2, \ldots, a_n)$, solve $a_1x_1 + by = c$, and iterate with $a_2x_2 + \cdots = y$.

Extended GCD

```
// Finds g = gcd(a,b) and x, y such that ax+by=g.
// Bounds: |x|<=b+1, |y|<=a+1.
void gcdext(int &g, int &x, int &y, int a, int b)
{ if (b == 0) { g = a; x = 1; y = 0; }
  else        { gcdext(g, y, x, b, a % b); y = y - (a / b) * x; } }
```

Multiplicative inverse of a modulo m: x in $ax + my = 1$, or $a^{\phi(m)-1} \pmod{m}$.

Chinese Remainder Theorem. System $x \equiv a_i \pmod{m_i}$ for $i = 1, \ldots, n$, with pairwise relatively-prime m_i has a unique solution modulo $M = m_1m_2\ldots m_n$: $x = a_1b_1\frac{M}{m_1} + \cdots + a_nb_n\frac{M}{m_n} \pmod{M}$, where b_i is modular inverse of $\frac{M}{m_i}$ modulo m_i.

System $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$ has solutions iff $a \equiv b \pmod{g}$, where $g = \gcd(m,n)$. The solution is unique modulo $L = \frac{mn}{g}$, and equals: $x \equiv a + T(b-a)m/g \equiv b + S(a-b)n/g \pmod{L}$, where S and T are integer solutions of $mT + nS = \gcd(m,n)$.

Prime-counting function. $\pi(n) = |\{p \leq n : p \text{ is prime}\}|$. $n/\ln(n) < \pi(n) < 1.3n/\ln(n)$. $\pi(1000) = 168$, $\pi(10^6) = 78498$, $\pi(10^9) = 50\ 847\ 534$. n-th prime $\approx n\ln n$.

Miller-Rabin's primality test. Given $n = 2^rs + 1$ with odd s, and a random integer $1 < a < n$.
If $a^s \equiv 1 \pmod{n}$ or $a^{2^js} \equiv -1 \pmod{n}$ for some $0 \leq j \leq r-1$, then n is a probable prime. With bases 2, 7 and 61, the test indentifies all composites below 2^{32}. Probability of failure for a random a is at most 1/4.

Pollard-ρ. Choose random x_1, and let $x_{i+1} = x_i^2 - 1 \pmod{n}$. Test $\gcd(n, x_{2^k+i} - x_{2^k})$ as possible n's factors for $k = 0, 1, \ldots$ Expected time to find a factor: $O(\sqrt{m})$, where m is smallest prime power in n's factorization. That's $O(n^{1/4})$ if you check $n = p^k$ as a special case before factorization.

Fermat primes. A Fermat prime is a prime of form $2^{2^n} + 1$. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form $2^n + 1$ is prime only if it is a Fermat prime.

Fermat's Theorem. Let m be a prime and x and m coprimes, then:

- $x^{m-1} \equiv 1 \mod m$
- $x^k \mod m = x^{k \mod (m-1)} \mod m$
- $x^{\phi(m)} \equiv 1 \mod m$

Perfect numbers. $n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

Carmichael numbers. A positive composite n is a Carmichael number ($a^{n-1} \equiv 1 \pmod{n}$ for all a: $\gcd(a,n) = 1$), iff n is square-free, and for all prime divisors p of n, $p - 1$ divides $n - 1$.

Number/sum of divisors. $\tau(p_1^{a_1} \ldots p_k^{a_k}) = \prod_{j=1}^{k}(a_j + 1)$. $\sigma(p_1^{a_1} \ldots p_k^{a_k}) = \prod_{j=1}^{k} \frac{p_j^{a_j+1} - 1}{p_j - 1}$.

Product of divisors. $\mu(n) = n^{\frac{\tau(n)}{2}}$

- if p is a prime, then: $\mu(p^k) = p^{\frac{k(k+1)}{2}}$
- if a and b are coprimes, then: $\mu(ab) = \mu(a)^{\tau(b)}\mu(b)^{\tau(a)}$

Euler's phi function. $\phi(n) = |\{m \in \mathbb{N}, m \leq n, \gcd(m,n) = 1\}|$.

- $\phi(mn) = \frac{\phi(m)\phi(n)\gcd(m,n)}{\phi(\gcd(m,n))}$.
- $\phi(p) = p - 1$ si p es primo
- $\phi(p^a) = p^a(1 - \frac{1}{p}) = p^{a-1}(p-1)$
- $\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})...(1 - \frac{1}{p_k})$ donde p_i es primo y divide a n

Euler's theorem. $a^{\phi(n)} \equiv 1 \pmod{n}$, if $\gcd(a,n) = 1$.
Wilson's theorem. p is prime iff $(p-1)! \equiv -1 \pmod{p}$.

Mobius function. $\mu(1) = 1$. $\mu(n) = 0$, if n is not squarefree. $\mu(n) = (-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. If for all $n \in N$, $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d)F(\frac{n}{d})$, and vice versa. $\phi(n) = \sum_{d|n} \mu(d)\frac{n}{d}$. $\sum_{d|n} \mu(d) = 1$.
If f is multiplicative, then $\sum_{d|n} \mu(d)f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) =$

$\prod_{p|n}(1 + f(p))$.
$\sum_{d|n} \mu(d) = e(n) = [n == 1]$.
$S_f(n) = \prod_{p=1}(1 + f(p_i) + f(p_i^2) + ... + f(p_i^{e_i}))$, p - primes(n).

Legendre symbol. If p is an odd prime, $a \in \mathbb{Z}$, then $\left(\frac{a}{p}\right)$ equals 0, if $p|a$; 1 if a is a quadratic residue modulo p; and -1 otherwise. Euler's criterion: $\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \pmod{p}$.

Jacobi symbol. If $n = p_1^{a_1} \cdots p_k^{a_k}$ is odd, then $\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{k_i}$.

Primitive roots. If the order of g modulo m (min $n > 0$: $g^n \equiv 1 \pmod{m}$) is $\phi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i = \text{ind}_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod{m}$. $\text{ind}_g(a)$ has logarithm-like properties: $\text{ind}(1) = 0$, $\text{ind}(ab) = \text{ind}(a) + \text{ind}(b)$.

If p is prime and a is not divisible by p, then congruence $x^n \equiv a \pmod{p}$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod{p}$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i \equiv a \pmod{p}$, $g^u \equiv x \pmod{p}$. $x^n \equiv a \pmod{p}$ iff $g^{nu} \equiv g^i \pmod{p}$ iff $nu \equiv i \pmod{p}$.)

Discrete logarithm problem. Find x from $a^x \equiv b \pmod{m}$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n = \lceil\sqrt{m}\rceil$, and $x = ny - z$. Equation becomes $a^{ny} \equiv ba^z \pmod{m}$. Precompute all values that the RHS can take for $z = 0, 1, \ldots, n-1$, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

Pythagorean triples. Integer solutions of $x^2 + y^2 = z^2$ All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n, \gcd(m, n) = 1$ and $m \not\equiv n \pmod{2}$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

- Given an arbitrary pair of integers m and n with $m > n > 0$:
  $a = m^2 - n^2$, $b = 2mn$, $c = m^2 + n^2$
- The triple generated by Euclid's formula is primitive if and only if m and n are coprime and not both odd.
- To generate all Pythagorean triples uniquely:
  $a = k(m^2 - n^2)$, $b = k(2mn)$, $c = k(m^2 + n^2)$
- If m and n are two odd integer such that $m > n$, then:
  $a = mn$, $b = \frac{m^2-n^2}{2}$, $c = \frac{m^2+n^2}{2}$
- If $n = 1$ or 2 there are no solutions. Otherwise
  n is even: $((\frac{n^2}{4} - 1)^2 + n^2 = (\frac{n^2}{4} + 1)^2)$
  n is odd: $((\frac{n^2-1}{2})^2 + n^2 = (\frac{n^2+1}{2})^2)$

Postage stamps/McNuggets problem. Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax + by$ ($x, y \geq 0$), and the largest is $(a-1)(b-1) - 1 = ab - a - b$.

Fermat's two-squares theorem. Odd prime p can be represented as a sum of two squares iff $p \equiv 1 \pmod{4}$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k + 3$ occurs an even number of times in n's factorization.

RSA. Let p and q be random distinct large primes, $n = pq$. Choose a small odd integer e, relatively prime to $\phi(n) = (p-1)(q-1)$, and let $d = e^{-1} \pmod{\phi(n)}$. Pairs (e, n) and (d, n) are the public and secret keys, respectively. Encryption is done by raising a message $M \in Z_n$ to the power e or d, modulo n.

String Algorithms

Burrows-Wheeler inverse transform. Let $B[1..n]$ be the input (last column of sorted matrix of string's rotations.) Get the first column, $A[1..n]$, by sorting B. For each k-th occurence of a character c at index i in A, let $next[i]$ be the index of corresponding k-th occurence of c in B. The r-th fow of the matrix is $A[r]$, $A[next[r]]$, $A[next[next[r]]]$, ...

Huffman's algorithm. Start with a forest, consisting of isolated vertices. Repeatedly merge two trees with the lowest weights.

Graph Theory

Euler's theorem. For any planar graph, $V - E + F = 1 + C$, where V is the number of graph's vertices, E is the number of edges, F is the number of faces in graph's planar drawing, and C is the number of connected components. Corollary: $V - E + F = 2$ for a 3D polyhedron.

Vertex covers and independent sets. Let M, C, I be a max matching, a min vertex cover, and a max independent set. Then $|M| \leq |C| = N - |I|$, with equality for bipartite graphs. Complement of an MVC is always a MIS, and vice versa. Given a bipartite graph with partitions (A, B), build a network: connect source to A, and B to sink with edges of capacities, equal to the corresponding nodes' weights, or 1 in the unweighted case. Set capacities of the original graph's edges to the infinity. Let (S, T) be a minimum s-t cut. Then a maximum(-weighted) independent set is $I = (A \cap S) \cup (B \cap T)$, and a minimum(-weighted) vertex cover is $C = (A \cap T) \cup (B \cap S)$.

Matrix-tree theorem. Let matrix $T = [t_{ij}]$, where t_{ij} is the number of multiedges between i and j, for $i \neq j$, and $t_{ii} = -\text{deg}_i$. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T.

Euler tours. Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected. Construction:

```
doit(u):
  for each edge e = (u, v) in E, do: erase e, doit(v)
  prepend u to the list of vertices in the tour
```

Stable marriages problem. While there is a free man m: let w be the most-preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

Stoer-Wagner's min-cut algorithm. Start from a set A containing an arbitrary vertex. While $A \neq V$, add to A the most tightly connected vertex ($z \notin A$ such that $\sum_{x \in A} w(x, z)$ is maximized.) Store cut-of-the-phase (the cut between the last added vertex and rest of the graph), and merge the two vertices added last. Repeat until the graph is contracted to a single vertex. Minimum cut is one of the cuts-of-the-phase.

Tarjan's offline LCA algorithm. (Based on DFS and union-find structure.)

```
DFS(x):
  ancestor[Find(x)] = x
  for all children y of x:
     DFS(y); Union(x, y); ancestor[Find(x)] = x
  seen[x] = true
  for all queries {x, y}:
    if seen[y] then output "LCA(x, y) is ancestor[Find(y)]"
```

Strongly-connected components. Kosaraju's algorithm.
1. Let G^T be a transpose G (graph with reversed edges.)
1. Call DFS(G^T) to compute finishing times $f[u]$ for each vertex u.
3. For each vertex u, in the order of decreasing $f[u]$, perform DFS(G, u).
4. Each tree in the 3rd step's DFS forest is a separate SCC.

2-SAT. Build an implication graph with 2 vertices for each variable – for the variable and its inverse; for each clause $x \vee y$ add edges $(\overline{x}, y)$ and $(\overline{y}, x)$. The formula is satisfiable iff x and $\overline{x}$ are in distinct SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (i.e. Kosaraju's last step), assigning 'true' to all variables of the current SCC (if it hasn't been previously assigned 'false'), and 'false' to all inverses.

Randomized algorithm for non-bipartite matching. Let G be a simple undirected graph with even $|V(G)|$. Build a matrix A, which for each edge $(u, v) \in E(G)$ has $A_{i,j} = x_{i,j}$, $A_{j,i} = -x_{i,j}$, and is zero elsewhere. Tutte's theorem: G has a perfect matching iff $\det G$ (a multivariate polynomial) is identically zero. Testing the latter can be done by computing the determinant for a few random values of $x_{i,j}$'s over some field. (e.g. Z_p for a sufficiently large prime p)

Prufer code of a tree. Label vertices with integers 1 to n. Repeatedly remove the leaf with the smallest label, and output its only neighbor's label, until only one edge remains. The sequence has length $n - 2$. Two isomorphic trees have the same sequence, and every sequence of integers from 1 and n corresponds to a tree. Corollary: the number of labelled trees with n vertices is n^{n-2}.

Erdos-Gallai theorem. A sequence of integers $\{d_1, d_2, \dots, d_n\}$, with $n - 1 \geq d_1 \geq d_2 \geq \dots \geq d_n \geq 0$ is a degree sequence of some undirected simple graph iff $\sum d_i$ is even and $d_1 + \dots + d_k \leq k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i)$ for all $k = 1, 2, \dots, n-1$.

Games

Grundy numbers. For a two-player, normal-play (last to move wins) game on a graph (V, E): $G(x) = \text{mex}(\{G(y) : (x, y) \in E\})$, where $\text{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

Sums of games.

- *Player chooses a game and makes a move in it.* Grundy number of a position is xor of grundy numbers of positions in summed games.
- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them.* A position is losing iff each game is in a losing position.
- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.
- *Player must move in all games, and loses if can't move in some game.* A position is losing if any of the games is in a losing position.

Misère Nim. A position with pile sizes $a_1, a_2, \dots, a_n \geq 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \dots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*.

Bit tricks

Clearing the lowest 1 bit: `x & (x - 1)`, all trailing 1's: `x & (x + 1)`
Setting the lowest 0 bit: `x | (x + 1)`
Enumerating subsets of a bitmask m:
`x=0; do { ...; x=(x+1+~m)&m; } while (x!=0);`
`__builtin_ctz`/`__builtin_clz` returns the number of trailing/leading zero bits.
`__builtin_popcount(unsigned x)` counts 1-bits (slower than table lookups).
For 64-bit unsigned integer type, use the suffix '`ll`', i.e. `__builtin_popcountll`.
XOR Let's say F(L,R) is XOR of subarray from L to R.
Here we use the property that F(L,R)=F(1,R) XOR F(1,L-1)

Math

Stirling's approximation $z! = \Gamma(z+1) = \sqrt{2\pi}\ z^{z+1/2}\ e^{-z}(1 + \frac{1}{12z} + \frac{1}{288z^2} - \frac{139}{51840z^3} + \dots)$

Taylor series. $f(x) = f(a) + \frac{x-a}{1!}f'(a) + \frac{(x-a)^2}{2!}f^{(2)}(a) + \dots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \dots$.
$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$
$\ln x = 2(a + \frac{a^3}{3} + \frac{a^5}{5} + \dots)$, where $a = \frac{x-1}{x+1}$. $\ln x^2 = 2\ln x$.
$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$, $\arctan x = \arctan c + \arctan \frac{x-c}{1+xc}$ (e.g c=.2)
$\pi = 4\arctan 1$, $\pi = 6\arcsin\frac{1}{2}$

Fibonacci Period Si p es primo , $\pi(p^k) = p^{k-1}\pi(p)$
$\pi(2) = 3$ $\pi(5) = 20$
Si n y m son coprimos $\pi(n*m) = lcm(\pi(n), \pi(m))$

List of Primes

1e5	3	19	43	49	57	69	103	109	129	151	153
1e6	33	37	39	81	99	117	121	133	171	183	
1e7	19	79	103	121	139	141	169	189	223	229	
1e8	7	39	49	73	81	123	127	183	213		

2-SAT Rules
$p \to q \equiv \neg p \vee q$
$p \to q \equiv \neg q \to \neg p$
$p \vee q \equiv \neg p \to q$
$p \wedge q \equiv \neg(p \to \neg q)$
$\neg(p \to q) \equiv p \wedge \neg q$
$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \wedge (q \to r) \equiv (p \wedge q) \to r$
$(p \to r) \vee (q \to r) \equiv (p \vee q) \to r$
$(p \wedge q) \vee (r \wedge s) \equiv (p \vee r) \wedge (p \vee s) \wedge (q \vee r) \wedge (q \vee s)$

Summations

- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=1}^{n} i^3 = (\frac{n(n+1)}{2})^2$
- $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
- $\sum_{i=1}^{n} i^5 = \frac{(n(n+1))^2(2n^2+2n-1)}{12}$
- $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ para $x \neq 1$

Compound Interest

- N is the initial population, it grows at a rate of R. So, after X years the popularion will be $N \times (1+R)^X$

Great circle distance or geographical distance

- d = great distance, ϕ = latitude, λ = longitude, Δ = difference (all the values in radians)
- σ = central angle, angle form for the two vector
- $d = r * \sigma$, $\sigma = 2 * \arcsin(\sqrt{\sin^2(\frac{\Delta\phi}{2}) + \cos(\phi_1)\cos(\phi_2)\sin^2(\frac{\Delta\lambda}{2})})$

Theorems

- There is always a prime between numbers n^2 and $(n+1)^2$, where n is any positive integer
- There is an infinite number of pairs of the from $\{p, p+2\}$ where both p and $p+2$ are primes.
- Every even integer greater than 2 can be expressed as the sum of two primes.
- Every integer greater than 2 can be written as the sum of three primes.
- $a^d \equiv a^{d \mod \phi(n)} \mod n$
  if $a \in \mathbb{Z}^{n*}$ or $a \notin \mathbb{Z}^{n*}$ and $d \mod \phi(n) \neq 0$
- $a^d \equiv a^{\phi(n)} \mod n$
  if $a \notin \mathbb{Z}^{n*}$ and $d \mod \phi(n) = 0$
- thus, for all a, n and d (with $d \geq \log_2(n)$)
  $a^d \equiv a^{\phi(n) + d \mod \phi(n)} \mod n$

Law of sines and cosines

- a, b, c: lenghts, A, B, C: opposite angles, d: circumcircle
- $\frac{a}{sin(A)} = \frac{b}{sin(B)} = \frac{c}{sin(C)} = d$
- $c^2 = a^2 + b^2 - 2ab\cos(C)$

Heron's Formula

- $s = \frac{a+b+c}{2}$
- $Area = \sqrt{s(s-a)(s-b)(s-c)}$
- a, b, c there are the lenghts of the sides

Legendre's Formula Largest power of k, x, such that $n!$ is divisible by k^x

- If k is prime, $x = \frac{n}{k} + \frac{n}{k^2} + \frac{n}{k^3} + \dots$

- If k is composite $k = k_1^{p_1} * k_2^{p_2} \dots k_m^{p_m}$
  $x = min_{1 \le j \le m} \{ \frac{a_j}{p_j} \}$ where a_j is Legendre's formula for k_j
- Divisor Formulas of n! Find all prime numbers $\le n$ $\{p_1, \dots, p_m\}$ Let's define e_j as Legendre's formula for p_j
- Number of divisors of n! The answer is $\prod_{j=1}^{m} (e_j + 1)$
- Sum of divisors of n! The answer is $\prod_{j=1}^{m} \frac{p_j^{e_j+1} - 1}{e_j - 1}$